UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

GRADO EN INGENERÍA EN ELECTRÓNICA Y AUTOMÁTICA INDUSTRIAL

Trabajo Fin de Grado

“Robot pintor artístico”

2014 Adrián Soto Garrido

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

GRADO EN INGENERÍA EN ELECTRÓNICA Y AUTOMÁTICA INDUSTRIAL

Trabajo Fin de Grado

"Robot pintor artístico"

Autor:	Adrián Soto Garrido
Universidad:	Universidad de Alcalá de Henares
País:	España
Profesor Tutor:	Rafael Barea Navarro

Presidente: D[a]. Elena López Guillén

Vocal 1: D. Luis Miguel Bergasa Pascual

Vocal 2: D. Rafael Barea Navarro

CALIFICACIÓN:.....

FECHA:.....

Agradecimientos

“Busca un grupo de gente que te rete e inspire, pasa mucho tiempo con ellos, y cambiará tu vida. Nadie está aquí hoy por haberlo hecho por sí solo”. (Amy Poehler).

Me gustaría agradecer a mi familia y amigos el apoyo recibido durante mi etapa en la universidad y por dejarme crecer a vuestro lado, sin vosotros mi camino no habría sido tan especial.

A mis compañeros y profesores, gracias por dejarme aprender a junto a vosotros y mostrarme la importancia de los valores que ofrece la universidad.

De manera especial, dar las gracias a Eva Ruiz a la que debo agradecer cada día de estos últimos 4 años, en ti encontré a la perfecta compañera para este viaje llamado vida.

Gracias por vuestra inspiración.

Contenido general

Lista de Figuras

Lista de tablas

Resumen

El objetivo de este proyecto es diseñar e implementar una aplicación, para el robot industrial ABB-IRB 120, capaz de transformar una imagen en información entendible para el robot, de tal forma que pueda representar la imagen original.

Esta aplicación desarrollada en distintos lenguajes de programación se divide en tres bloques, siendo el primer bloque el principal y en el que más se va a trabajar.

El primer bloque realizado con el software "Visual Studio", en lenguaje C#, tiene como objetivo la captura y procesamiento de imágenes a través de la cámara web.

El segundo bloque consta de una comunicación entre el software del robot y el primer bloque, aquí se realiza el intercambio de información entre ambos. Este segundo bloque está desarrollado tanto en lenguaje C# como en lenguaje RAPID.

Y por último el tercer bloque, cuya función será la interpretación de los datos recibidos a través del segundo bloque, de tal forma que el robot es capaz de representar la imagen original.

1 Introducción

1.1 Sociedad actual

Vivimos en época de permanente tránsito, de continua obsolescencia.

Conocemos varias definiciones de tecnología y todas ellas difieren unas de otras en pequeños matices debido a la imposibilidad de definir algo que está en permanente evolución. Lo que sí podemos decir de la tecnología son algunos de los efectos que tiene en la sociedad.

La actividad tecnológica compromete de manera sustancial a la sociedad en la que vivimos, su carácter abrumadoramente comercial nos dice que está diseñada para satisfacer los deseos de aquellos económicamente capaces y no de cubrir las necesidades de aquello que requiere de la actividad tecnológica. Este hecho provoca que las empresas tecnológicas tengan la innovación como argumento más importante para continuar en el mercado.

Este argumento que tan necesario ha sido, ha conducido en particular a la robótica a alcanzar niveles de sofisticación impensables hace 60 años, cuando se daban las primeras pinceladas de una obra que a día de hoy es mucho más completa, pero seguro inacabada.

Hoy en día, cuando hablamos de robótica, hablamos de una rama de la tecnología totalmente indispensable para la vida. La implementación de sistemas robóticos desde áreas domésticas hasta áreas aeroespaciales nos hace ver su tremendo potencial. Incluso otras ramas de la ciencia, como puede ser la medicina, están empezando a utilizar la robótica para perfeccionarse y quién sabe si en un futuro no llegue a ser totalmente necesario, de forma que se alcancen metas a día de hoy no vislumbradas.

Todas las aplicaciones que encontramos en la robótica suponen un sinfín de posibilidades que hace años no existían y que hoy son posibles para cualquier usuario. Uno de los factores por los cuales decimos que estamos en permanente evolución es precisamente acercar estas posibilidades a cualquier usuario. Este hecho se debe a que hoy en día cualquiera que posea el interés y los conocimientos adecuados de robótica es capaz de diseñar e implementar un sistema robótico.

Por tanto, tenemos un mar entero de posibilidades que permiten, a todo aquel que se decida a sumergirse en él, para experimentar, investigar, disfrutar y, porque no, hasta innovar. Completando así el gran ciclo de la actividad tecnológica.

1.2 Estado del arte

Como se ha mencionado anteriormente, existe un amplio abanico de posibilidades que permite a la robótica crecer en diferentes áreas como la automoción, la medicina o incluso el arte.

Estas aplicaciones tienen un fuerte impacto en la sociedad actual, ¿cómo es posible que una persona tarde años en dominar un determinado arte y un robot sólo necesite una pequeñísima fracción de este tiempo?

Algunas personas consideran que el arte de la pintura lleva años y años de práctica y aun así no puede llegar a ser dominado por todos. Sin embargo, un robot puede "aprender" este arte en cuestión de horas.

Actualmente existen diferentes empresas y desarrolladores trabajando en aplicaciones de este tipo.

Robots capaces de plasmar a la perfección una imagen captada por una cámara, como si del mejor de todos los artistas se tratase. Pintar con grafito, con rotulador o incluso pincel y témpera. No hay técnica que el robot no pueda desarrollar.

Compañías como ABB, que cuentan con una gran cantidad de brazos robóticos en su portfolio, han desarrollado aplicaciones de este tipo, como se puede ver en la siguiente imagen.

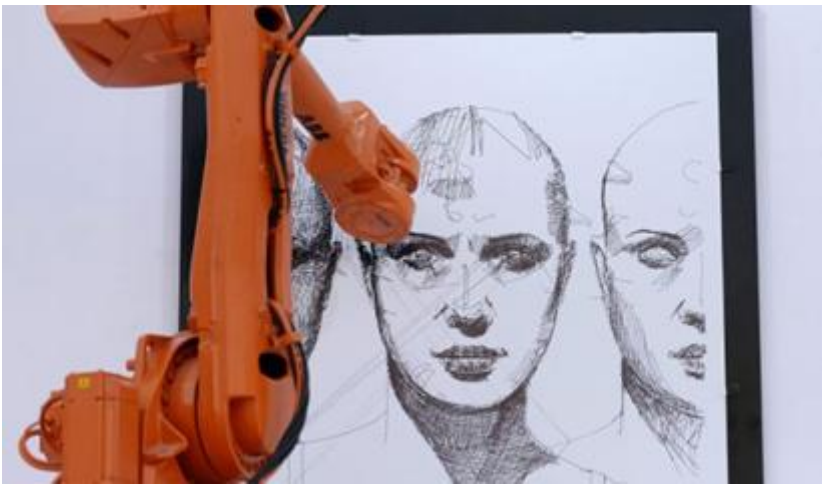


Figura 1.1 Robot ABB pintor

En la imagen se puede observar como el robot ha procesado una información que se le ha proporcionado, de manera que se puede apreciar una imagen, que era el objetivo de la aplicación.

Pero existen muchos otros prototipos de aplicaciones que pueden ser implementadas en brazos robóticos para este fin. Por ejemplo se puede citar al ya bautizado "e David".

e David es un robot creado por científicos en la universidad de Konstanz, Alemania, y su nombre es el acrónimo de “Drawing Apparatus for Vivid Interactive Display”.

Este robot, es un robot de soldadura ordinario, que normalmente se utilizaría en la fabricación de coches. Pero combinando este robot con distintos sensores, una cámara y un software de control ha sido posible la implementación de una aplicación de dibujo muy interesante.

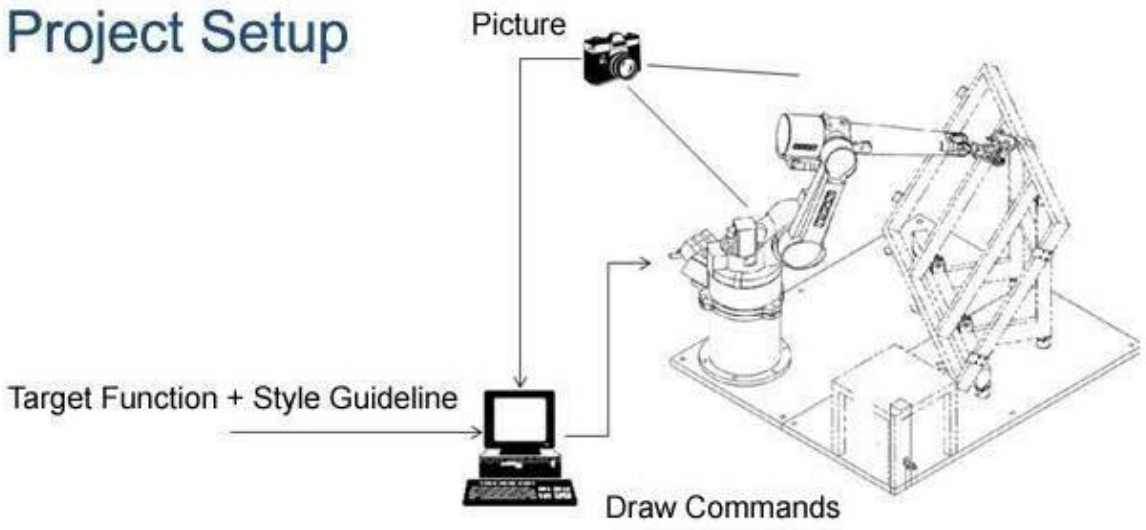


Figura 1.2 Arquitectura del sistema “e David”

En la imagen se aprecia como gracias a las imágenes adquiridas por la cámara y a las instrucciones pasadas por el usuario, a través del ordenador, este consigue mandar al robot una serie de comandos de dibujo con los que representa la imagen deseada.

El robot, además, cuenta con una herramienta pincel y una paleta de 24 colores. También posee una cámara capaz de monitorear todos los movimientos, mediante un feedback de un lazo de control del pincel y ajustar los colores para finalmente plasmar la imagen.

Su potencial ha hecho que, poco a poco, fuera evolucionando. En una primera versión del e David sólo se incluía una herramienta rotulador, con la que se pintaban imágenes en niveles de gris. Con el paso del tiempo, se han hecho una serie de actualizaciones para que hoy en día reproduzca un cuadro de la manera más real posible.

En la imagen que se detalla a continuación se puede hacer una idea de la precisión del brazo robótico y la variedad de trazos que puede ejecutar, de forma que el dibujo adopte una mayor realidad. Además se puede distinguir en el efector final del robot un sensor. Este sensor es el encargado de monitorear la posición del pincel y realizar una calibración del sistema en tiempo real, si esto fuera necesario.

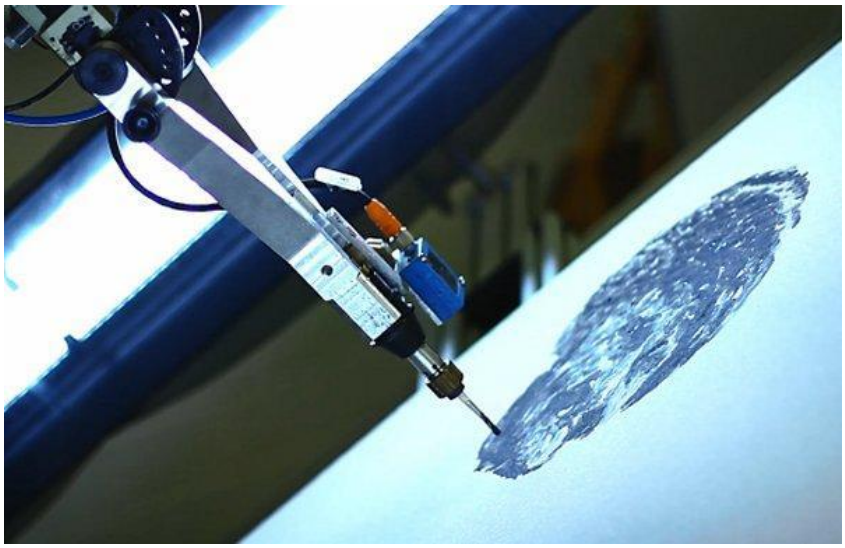


Figura 1.3 Ejemplo de dibujo con “e David”

El objetivo principal de este proyecto es realizar una aplicación que se asemeje de cierta manera al robot e David. Se dice de cierta manera porque, no se poseen todas las herramientas necesarias para implementar un proyecto de tal nivel de complejidad. Lo que sí se puede realizar es, como el e David en su primera versión, una aplicación que pinte una imagen en niveles de gris.

De esta manera queda definido el fin del proyecto: la realización de una aplicación capaz de representar unas determinadas imágenes en niveles de gris.

2 Herramientas utilizadas

En este punto se realizará un breve resumen acerca de las herramientas, tanto software como hardware, utilizadas para la realización de la aplicación. Estas herramientas han sido seleccionadas de manera objetiva con la intención de garantizar una mayor calidad en el proyecto.

2.1 Microsoft Visual Studio

Visual Studio.NET es un conjunto de herramientas de desarrollo para la construcción de aplicaciones.

Soporta múltiples lenguajes de programación como C++, Java, C#, Visual Basic.NET, F#, Python, Ruby, PHP; así como entornos de desarrollo web como ASP.NET o Django entre otros.

Este software permite a los desarrolladores crear aplicaciones, aplicaciones web, y servicios web soportables por la plataforma .NET. De esta manera, se pueden desarrollar aplicaciones que se intercomuniquen entre estaciones de trabajo, dispositivos móviles y páginas web.

Muchas son las versiones de Visual Studio desde su lanzamiento en 1997, pero el cambio más significativo se da en el año 2002. Este año supuso la introducción de la plataforma .NET de Microsoft. Se trata de una plataforma de ejecución intermedia multilenguaje, lo que significa que los programas que se desarrollan en esta plataforma se compilan en un lenguaje intermedio y no en lenguaje máquina.

En las aplicaciones realizadas en este formato, el código no se convierte a lenguaje máquina hasta que se ejecuta, con lo que el código puede ser totalmente independiente de plataforma.

Actualmente la última versión estable del software es Visual Studio 2013, y es el software utilizado en el desarrollo de más de la mitad de la aplicación que tiene como objeto este proyecto.

2.1.1 Plataforma .NET

Es conveniente entrar en más detalle de todo el potencial de esta plataforma.

.NET conduce a la tercera generación de Internet. La primera generación se caracterizó por trabajar con información estática que podía ser consultada a través de exploradores. La segunda generación consistía en que las aplicaciones pudieran interactuar con las personas. Y por último, la tercera generación, se basa en aplicaciones capaces de interactuar con otras aplicaciones.

Precisamente, el principio de .NET es que los sitios Web aislados y los diferentes dispositivos trabajen conectados a través de internet. Esto se consigue gracias a la aceptación de los estándares basados en XML (Extensible Markup Language).

Microsoft .NET extiende las ideas de Internet y sistema operativo haciendo de la propia Internet la base de un nuevo sistema operativo. Esto permite a los desarrolladores crear programas que aprovechen al máximo los dispositivos así como la conectividad de la red y sus aplicaciones.

Para ello proporciona una plataforma que incluye los siguientes componentes:

- Un conjunto de servicios que actúan como bloques de construcción para el sistema operativo de Internet

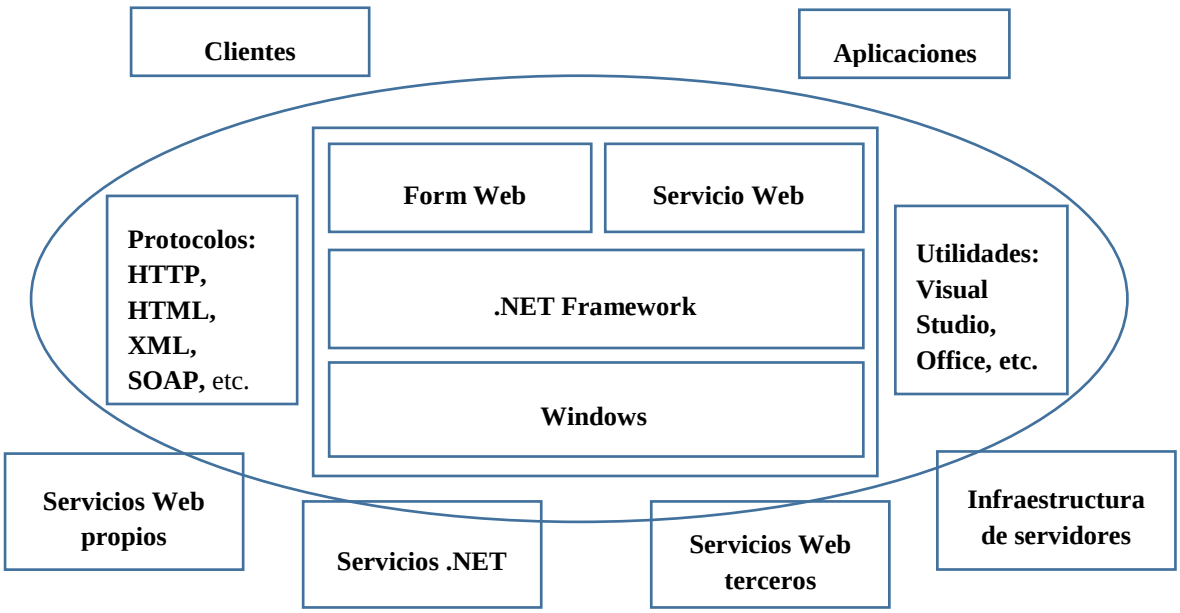


Figura 2.1 Bloques de la plataforma .NET

- Software de dispositivos .NET
- Herramientas de programación para crear servicios Web XML con soporte multilingüe: Visual Studio.NET y .NET Framework
- Infraestructura de servidores, incluyendo Windows y .NET Enterprise Servers.

Pero también se requiere de una infraestructura, no solo para facilitar el desarrollo de aplicaciones, sino también para hacer que el proceso de encontrar un servicio Web e integrarlo en una aplicación resulte transparente para usuarios y desarrolladores. Esta estructura nos la proporciona .NET Framework:

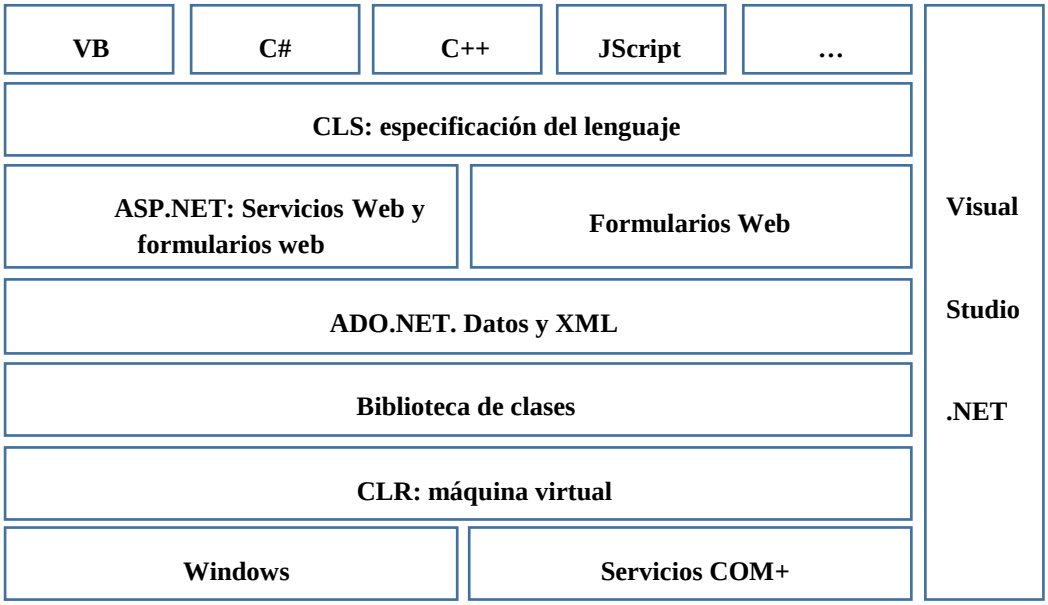


Figura 2.2 Estructura de la plataforma .NET

.NET Framework proporciona un entorno unificado para todos los lenguajes de programación.

Microsoft ha incluido en este marco los lenguajes Visual Basic, C#, C++ y JavaScript. Lo más llamativo quizá sea la posibilidad de escribir parte de una aplicación en un C# y la otra parte en Visual Basic. Sin embargo para que esto pueda ocurrir se propone la especificación común para los lenguajes (CLS). Para eso, define un conjunto de tipos de datos comunes que indican que tipos de datos se pueden manejar, la forma de declararlos y de utilizarlos.

Así aunque cada lenguaje .NET use un sintaxis diferente para cada tipo de datos, por ejemplo en VB se utiliza Integer para un número entero de 32 bits y en C# se utiliza Int, estos nombres no son más que sinónimos para el tipo común System.Int32. De esta manera se permite la interoperabilidad entre lenguajes.

.NET Framework proporciona un entorno de ejecución llamado CLR (Common Language Rutine). Se trata de una máquina virtual que administra la ejecución del código y proporciona servicios que hacen más cómodo el desarrollo.

Cuando se compila el código escrito, el compilador lo traduce a un código intermedio denominado MSIL (Microsoft Intermediate Language). Esto supone que cualquier lenguaje .NET puede implementarse en cualquier plataforma (Intel, Motorola…) que tenga instalada una máquina virtual .NET.

Esquema orientativo:

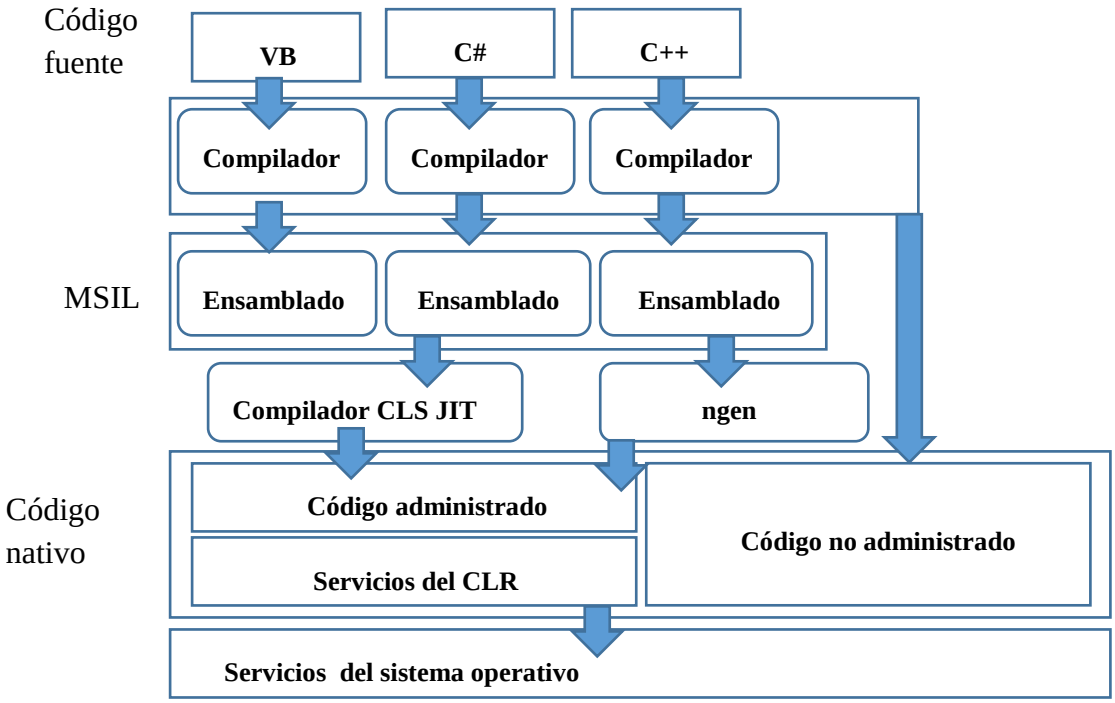


Figura 2.3 Esquema orientativo de uso de la Plataforma .NET

2.1.2 Lenguaje C#

El lenguaje C# es un lenguaje de programación orientado a objetos y ha sido diseñado por Microsoft en la iniciativa .NET.

Es sencillo y sigue el mismo patrón de los lenguajes de programación modernos. Incluye un amplio soporte de estructuras, manipulación de objetos…. Las clases son la base de los lenguajes de programación orientado a objetos, y C# contiene las herramientas para definir nuevas clases, sus propiedades y sus métodos, así como la capacidad de implementar encapsulación, herencia y polimorfismo que son los tres pilares de la programación orientada a objetos.

C# posee un nuevo estilo de documentación XML que se añade a lo largo de la aplicación, también provee soporte para estructuras.

Pero, ¿por qué C#? La razón fundamental es que se diseñó para la plataforma .NET y es capaz de aprovechar todo su potencial. También añadir que es un lenguaje "limpio" en el sentido de que, al no tener que proporcionar compatibilidad hacia etapas anteriores, se ha dedicado más tiempo al diseño y se ha hecho especial hincapié en su simplicidad.

2.2 Compañía ABB

ABB es una empresa líder mundial en ingeniería eléctrica y automatización.

La compañía es el producto de la unión en 1988 de Asea y BBC. Actualmente la sede central se encuentra en Zúrich, Suiza, cuenta con más de 150.000 empleados y está presente en más de 100 países.

Tiene una gran oferta en la que se puede diferenciar cinco divisiones: Power Products, Power Systems, Discrete Automation and Motion y Low Voltage Products.

En estas divisiones se ofrecen desde interruptores de iluminación, hasta robots industriales, grandes transformadores eléctricos o sistemas de control capaces de gestionar grandes redes eléctricas o industrias.

Como se ha mencionado, ABB suministra robots industriales con los que se mejora la productividad de las empresas, cuenta con una oferta de más de 25 tipos de robots distintos para las diferentes utilidades que se le puede dar en cada tipo de industria.

Este gran abanico de posibilidades que ofrece ABB le convierte en la empresa líder de suministro de robots en el mundo.

Además este suministro de robots se incrementa cada año debido a la gran necesidad de las empresas de evolucionar hacia una eficiencia energética y tecnológica, que les mantenga dentro del mercado.

Figura 2.4 Cadena de producción robots ABB

En la imagen se puede ver como toda una cadena de montaje de coches está llevada a cabo por robots de ABB. Se puede vislumbrar la importancia de los robots en este tipo de sistemas.

2.2.1 Software Robot Studio

RobotStudio es el software de simulación y programación offline que permite crear y simular estaciones robóticas industriales en 3D en un ordenador. Sus características aumentan la variedad de tareas posibles para llevar a cabo mediante un sistema de robótico, como la capacitación, la programación o la optimización.

Algunos de ellos ofrecen la reducción de riesgos, una puesta en marcha más rápida o el cambio de formato más corto. Además, cuando se utiliza RobotStudio con los controladores reales, se conoce como "modo en línea".

El software está basado en el controlador virtual de ABB, que es, una copia simulada del software real que utilizan los robots de ABB. Gracias a este software se puede ejecutar en nuestro ordenador un sistema robótico que se haya diseñado antes de ejecutarlo en el robot real, con lo que se evitan costes innecesarios.

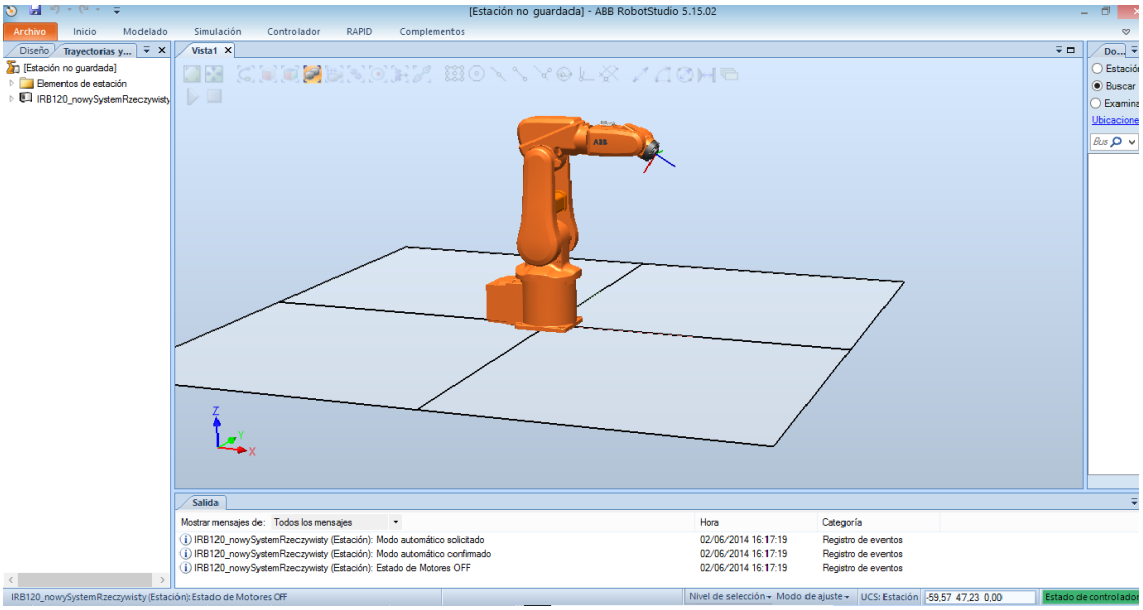


Figura 2.5 Interface principal Robtstudio

RobotStudio y su lenguaje de programación RAPID, tienen gran importancia en el desarrollo de este proyecto, pero toda la información necesaria del mismo, está completamente explicada y comentada en el proyecto "Socket based communication in RobotStudio for controlling ABB-IRB120 robot. Design and development of a palletizing station" de Marek Frydrysiak, por lo que no se entrará en explicaciones detalladas del software.

2.2.2 Brazo robótico IRB-120

Para llevar a cabo la parte final de la aplicación, se requiere un brazo robótico que ejecute la aplicación que se ha desarrollado. Este brazo robótico, de la compañía ABB, es el modelo IRB -120 y a continuación se detallan algunas de sus características principales.

Figura 2.6 Robot ABB IRB-120

Como se ha dicho, se dispone de un brazo robótico de tamaño reducido pero suficiente para el objetivo que se persigue. La estructura del brazo es la siguiente:

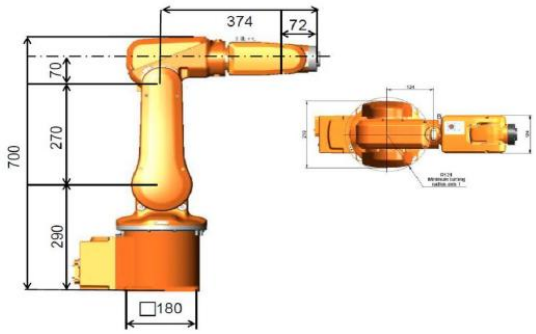


Figura 2.7 Dimensiones robot ABB IRB-120

Además posee 6 ejes, los tres primeros servirán para establecer la posición del efector final y los tres últimos para determinar la orientación del mismo.

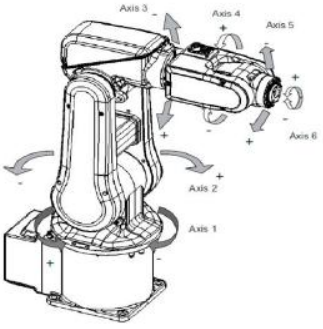


Figura 2.8 Articulaciones robot ABB IRB-120

2.3 Librerías utilizadas

Este capítulo tendrá varios objetivos. El primero de ellos será dar una noción sobre lo que es Open CV y su importancia en esta clase de sistemas. Se explicará, también, lo que es Emgu CV y la necesidad de su uso. Y por último se incluirá una guía de instalación para las librerías de Emgu CV, con las que el usuario podrá instalarlas.

2.3.1 Open CV

Open CV es una biblioteca libre de visión artificial creada en el año 1999 por Intel. En ella se pueden encontrar más de 500 funciones que abarcan un gran abanico de posibilidades, como el reconocimiento de objetos, reconocimiento facial, calibración de cámaras, visión estérea y visión robótica.

Su gran capacidad permite que se emplee en una gran cantidad de aplicaciones, desde sistemas de seguridad con detección de movimiento, hasta aplicativos de control de procesos donde se requiere reconocimiento de objetos. Esto, además, es gracias a que su publicación se da bajo licencia BSD, que permite que sea usada libremente para propósitos comerciales y de investigación.

Una de las limitaciones, es que esta librería no se puede implementar de forma directa en una aplicación de Windows Forms, escrita en lenguaje C# por lo que se necesita una solución práctica con la que se consiga poder hacer uso de esta librería tan potente

2.3.2 Descripción del problema

Mediante el software Visual Studio 2013 se realiza una aplicación de Windows Forms escrita en C#.

Como la aplicación final tendrá por objeto principal el procesamiento de imágenes, es recomendable la utilización de las funciones contenidas en la biblioteca “Open CV” ya que están diseñadas para tal propósito.

Como ya se ha mencionado, Open CV es una biblioteca de visión artificial donde se puede encontrar una serie de funciones que permiten una gran cantidad de posibilidades, como el reconocimiento de objetos, reconocimiento facial, calibración de cámaras, visión estérea y visión robótica.

El principal inconveniente es que no es posible implementar de forma directa las funciones de Open CV en la aplicación, con lo que hay que ayudarse de algún tipo de plataforma envoltorio con la cual, poder invocar a dichas funciones de manera directa.

2.3.3 Solución al problema

Una posible solución a este primer inconveniente, como se ha mencionado en el capítulo anterior, es implementar dichas funciones a través de la plataforma Emgu CV.

Emgu CV es una librería envoltorio que permite la implementación de las funciones contenidas en la librería de Open CV en lenguajes como C#, Visual Basic, Visual C ++, IronPython…

Esta plataforma puede compilarse en el compilador "Mono", por lo que se ejecuta en Windows, Linux, Mac OS X, iPhone, iPad y dispositivos Android.

Así, por tanto, se pueden utilizar sin problema alguno las funciones ya desarrolladas contenidas en Open CV en la realización de la aplicación de Windows Forms.

Figura 2.9 Librería Open CV y Emgu CV

2.3.4 Desarrollo y manual de instalación

Como se ha adelantado, para solucionar este problema se hace uso de la plataforma Emgu CV, su uso no es muy complejo y satisface todos los requerimientos necesarios para desarrollar la aplicación.

En primer lugar se debe descargar Emgu CV del sitio web (proporcionado en bibliografía – [4]) y seguir una serie de pasos con los que se podrá disponer de la plataforma sin dificultad.

Un primer paso, una vez descargada la librería, es copiar todos los archivos contenidos en la carpeta "\bin\x86" de la carpeta de instalación de Emgu CV, que son los siguientes:

cudart64_42_9.dll
cvextern.dll
npp64_42_9.dll
opencv_calib3dXXX.dll
opencv_contribXXX.dll
opencv_coreXXX.dll
opencv_features2dXXX.dll
opencv_flannXXX.dll
opencv_highguiXXX.dll
opencv_imgprocXXX.dll
opencv_legacyXXX.dll
opencv_mlXXX.dll
pencv_nonfreXXX.dll
opencv_objectdetectXXX.dll
opencv_videoXXX.dll

Tabla 2.1 Archivos .dll librería Emgu CV

A continuación, estos archivos deben pegarse en la carpeta en la que se encuentra el ejecutable del proyecto en cuestión, dentro de la carpeta "Debug". Por ejemplo dentro de la carpeta:

"C:\Users\XXX\Documents\VisualStudio2013\Projects\RobotDraw\RobotDraw\bin\Debug"

El segundo paso consiste en agregar las referencias necesarias.

Emgu.CV.dll
Emgu.CV.UI.dll
Emgu.Util.dll

Tabla 2.2 Referencias librería Emgu CV

Estas referencias se encuentran dentro de la carpeta BIN de la carpeta de instalación de Emgu CV y se agregan dentro de la propia aplicación de Windows Forms como cualquier otra referencia.

Una vez seguidos estos pasos, ya se pueden invocar a las funciones de Open CV por medio de Emgu CV, con lo que hay que tener en cuenta el funcionamiento y la estructura de las clases de Emgu CV para un correcto funcionamiento.

Además, para este caso en concreto, será necesario incluir, aparte de todo lo especificado, una serie de herramientas que también están contenidas en estas librerías. Estas herramientas son fundamentales en esta aplicación ya que en conjunto con las funciones se puede formar un sistema correcto en el que la aplicación funcione sin problemas.

Algunas de estas herramientas son conocidas como ImageBox, HistogramBox o MatrixBox.

Entrando más en detalle, una de las funcionalidades de la aplicación será visualizar por pantalla cualquier imagen obtenida por cualquier fuente que permite la aplicación. Para ello, se requiere una herramienta capaz de "contener" dichas imágenes. Esta herramienta es la conocida como ImageBox.

Para añadir esta herramienta, y las otras citadas anteriormente, se deben seguir una serie de pasos. En primer lugar, desde la vista diseño del Window Form, en la ventana ToolBox, se hace clic con el botón derecho del ratón en "General", al desplegarse el menú se hace clic en "Choose Items".

En segundo lugar, aparece un panel como el que se detalla en el que hay que añadir las librerías Emgu. Para ello, se hace clic en "Browse".

Primer Paso **Segundo Paso**

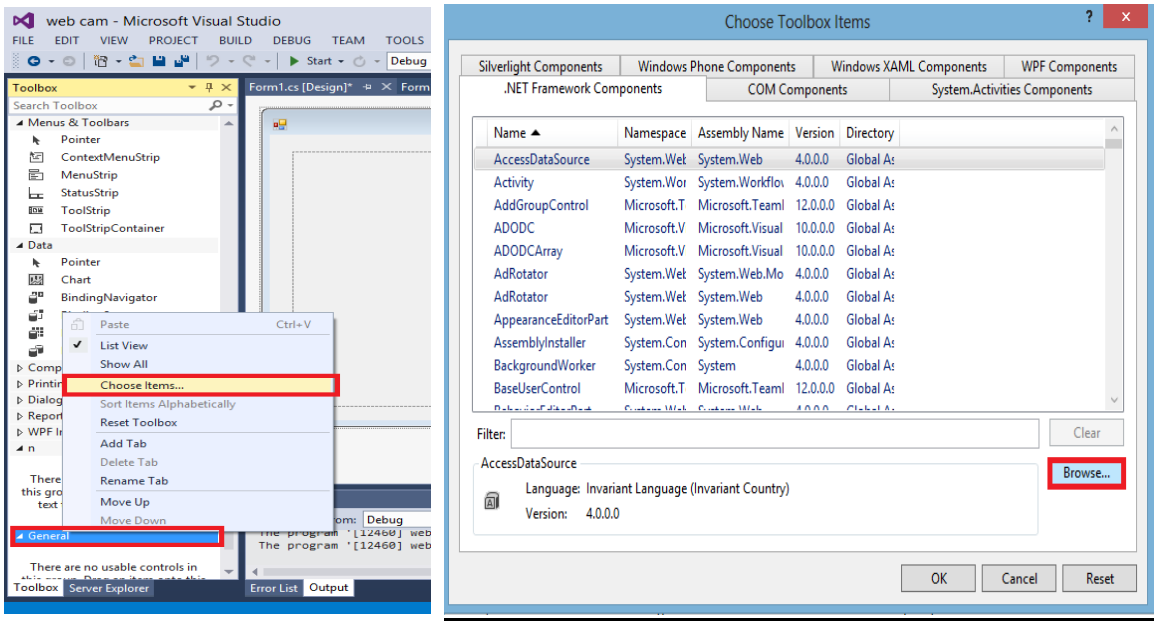


Figura 2.10 Primer paso y segundo paso de instalación de librerías

Una vez se haya hecho clic en "Browse" se selecciona "Emgu.CV.UI.dll", que se encuentran en la carpeta:

C:\[…]\VisualStudio2013\Projects\RobotDraw\RobotDraw\bin\Debug y se hace clic en "Abrir".

Tercer paso

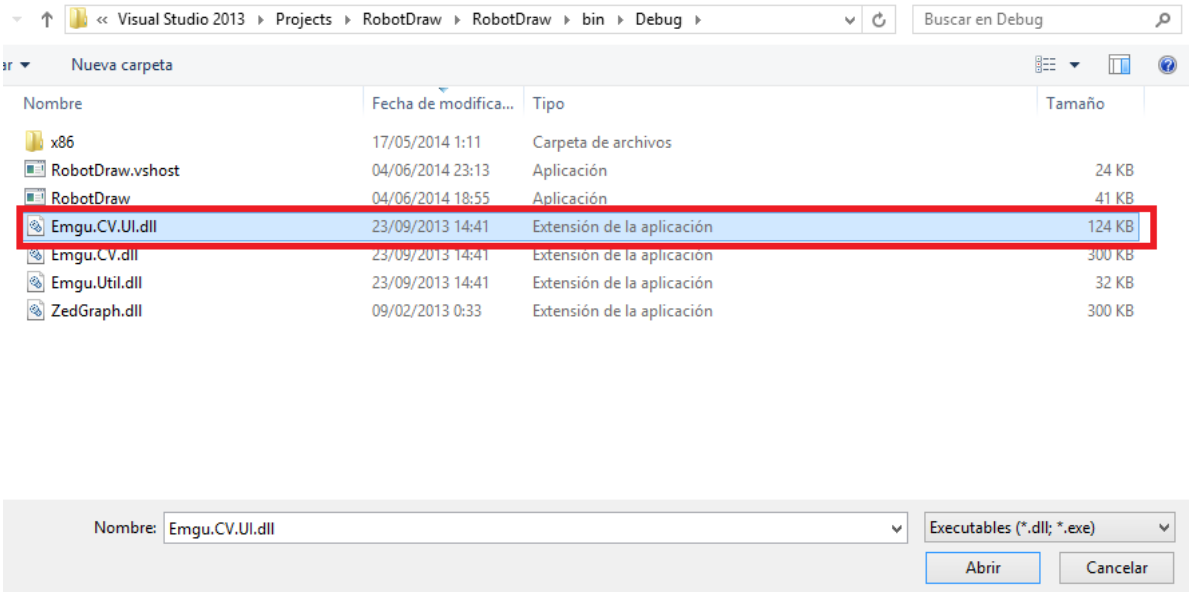


Figura 2.11 Tercer paso de instalación de librerías

Ya están incluidas las herramientas en el proyecto, ahora solo queda seleccionarlas para que sean visibles desde la interface principal, y que de esta manera se tenga un acceso directo a ellas.

Cuarto paso

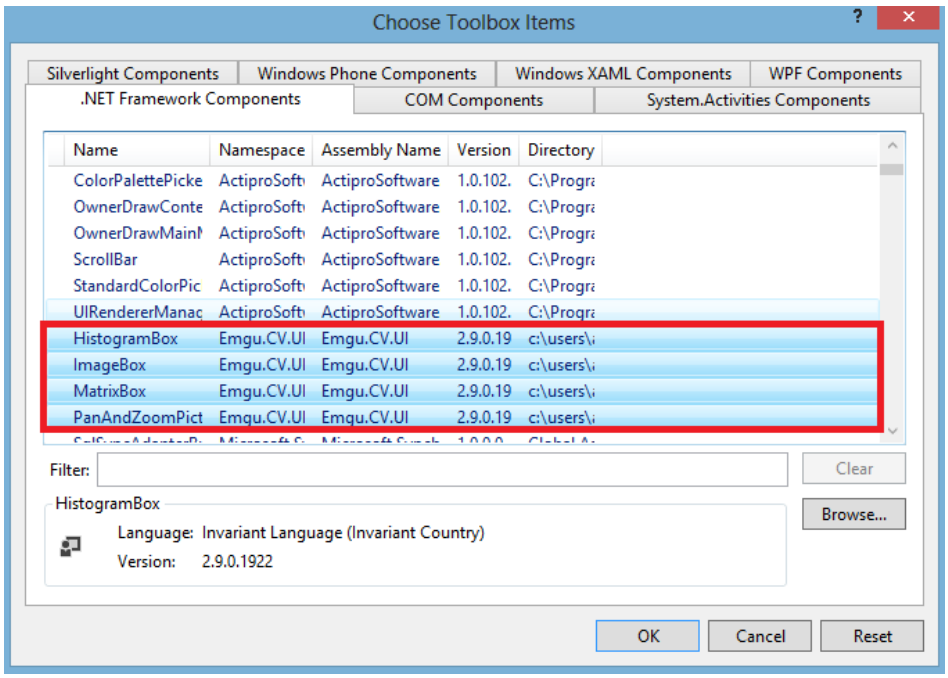


Figura 2.12 Cuarto paso de instalación de librerías

Así, ya se dispone de todas las herramientas y funciones necesarias para la implementación de la aplicación tal y como se ve en la siguiente imagen.

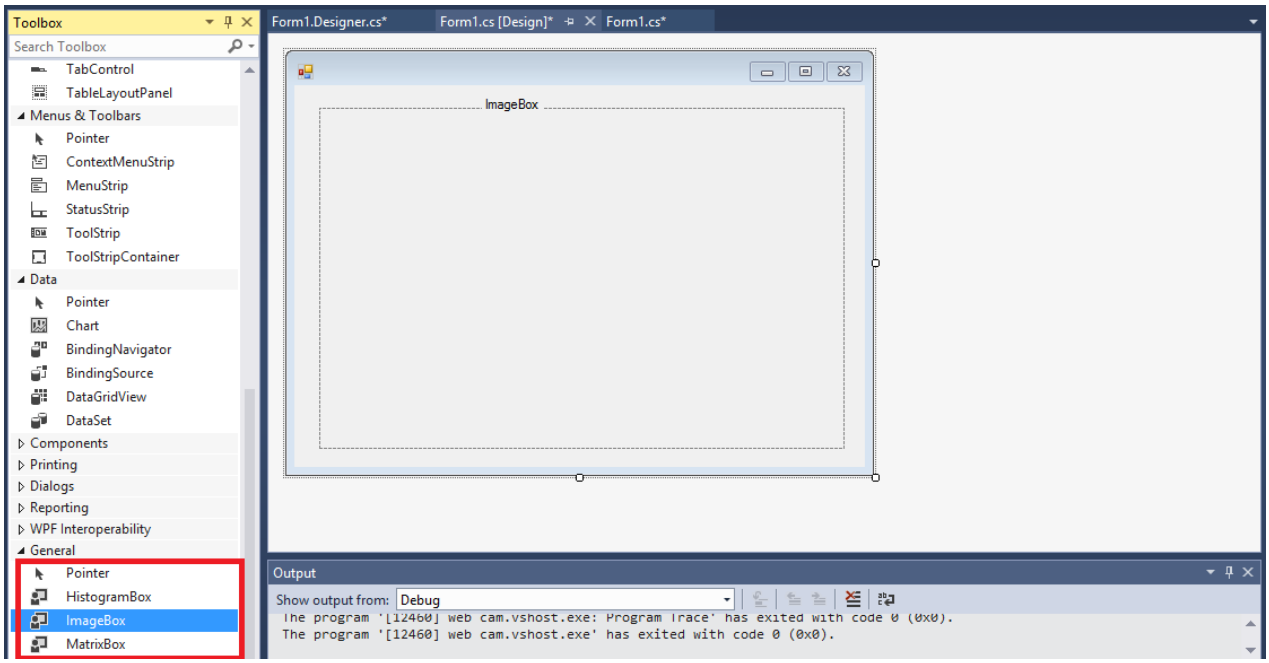


Figura 2.13 Inserción de herramientas en la aplicación

El único detalle que qeda por resaltar es, que para el uso de algunas funciones se debe agregar al comienzo del progama C una serie de bibliotecas.

La biblioteca de clases de .NET Framework es una biblioteca de clases, interfaces y tipos de valor que proporcionan acceso a la funcionalidad del sistema. Es la base sobre la que se compilan aplicaciones, componentes y controles de .NET Framework

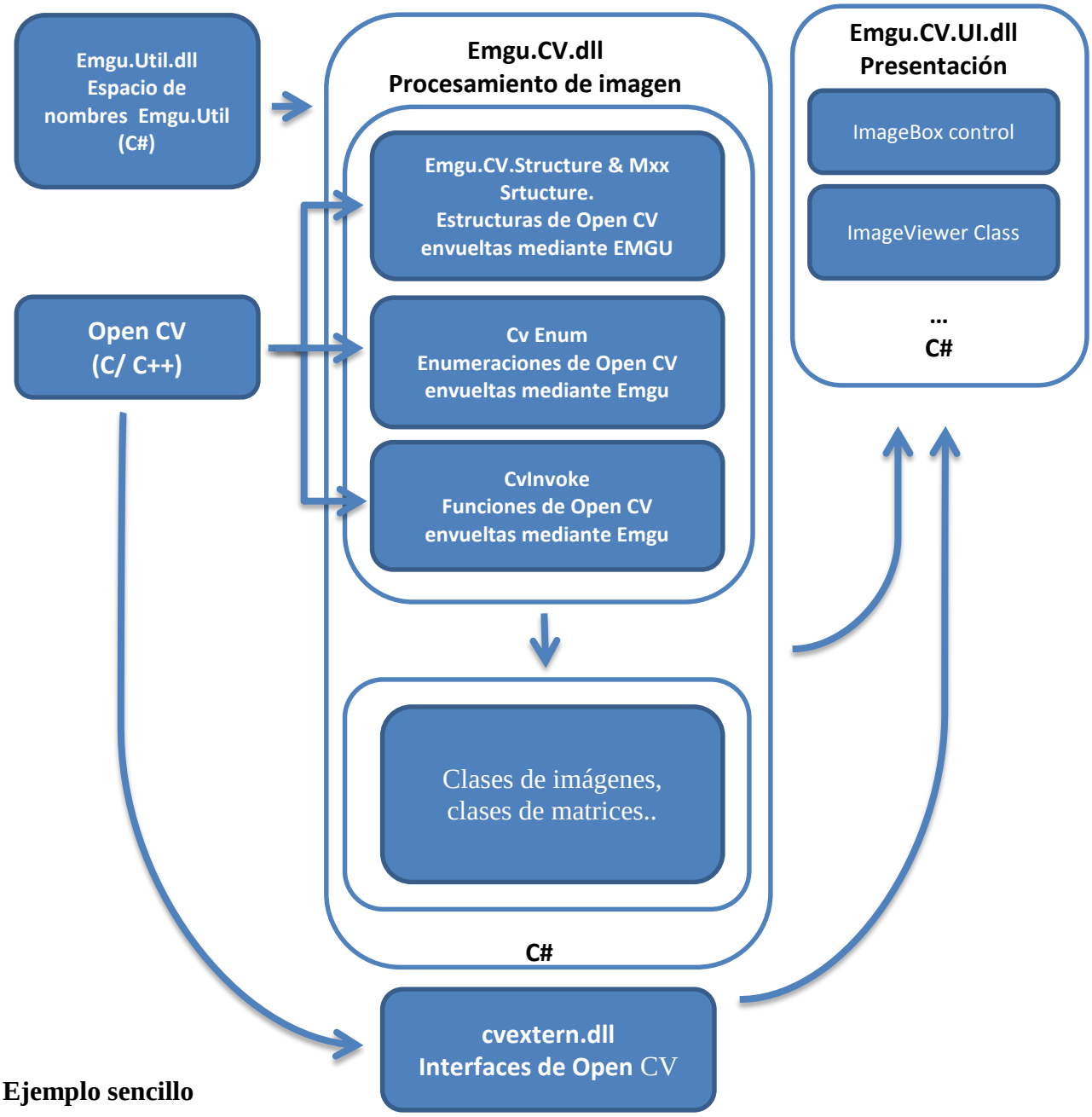
Para esta aplicación en particular se deben añadir:

```
using Emgu.CV;
using Emgu.CV.UI;
using Emgu.Util;
using Emgu.CV.Structure;
using Emgu.CV.CvEnum;
using System.Collections;
using System.Drawing.Drawing2D
using System.Threading;
```

Tabla 2.3 Bibliotecas utilizadas con Emgu CV

En la página siguiente se muestra un esquema orientativo del funcionamiento de Emgu CV y sus distintos niveles.

Esquema



Ejemplo sencillo

La clase CV Invoke proporciona una forma directa de llamar a la función de Open CV dentro de los lenguajes .NET.

```
IntPtr image = CvInvoke.cvCreateImage(new System.Drawing.Size(400, 300),
CvEnum.IPL_DEPTH.IPL_DEPTH_8U, 1);
```

Es equivalente a :

```
IplImage* image = cvCreateImage(cvSize(400, 300), IPL_DEPTH_8U, 1);
```

Figura 2.14 Esquema y ejemplo práctico Emgu CV

3 Arquitectura general del sistema

3.1 Esquema orientativo

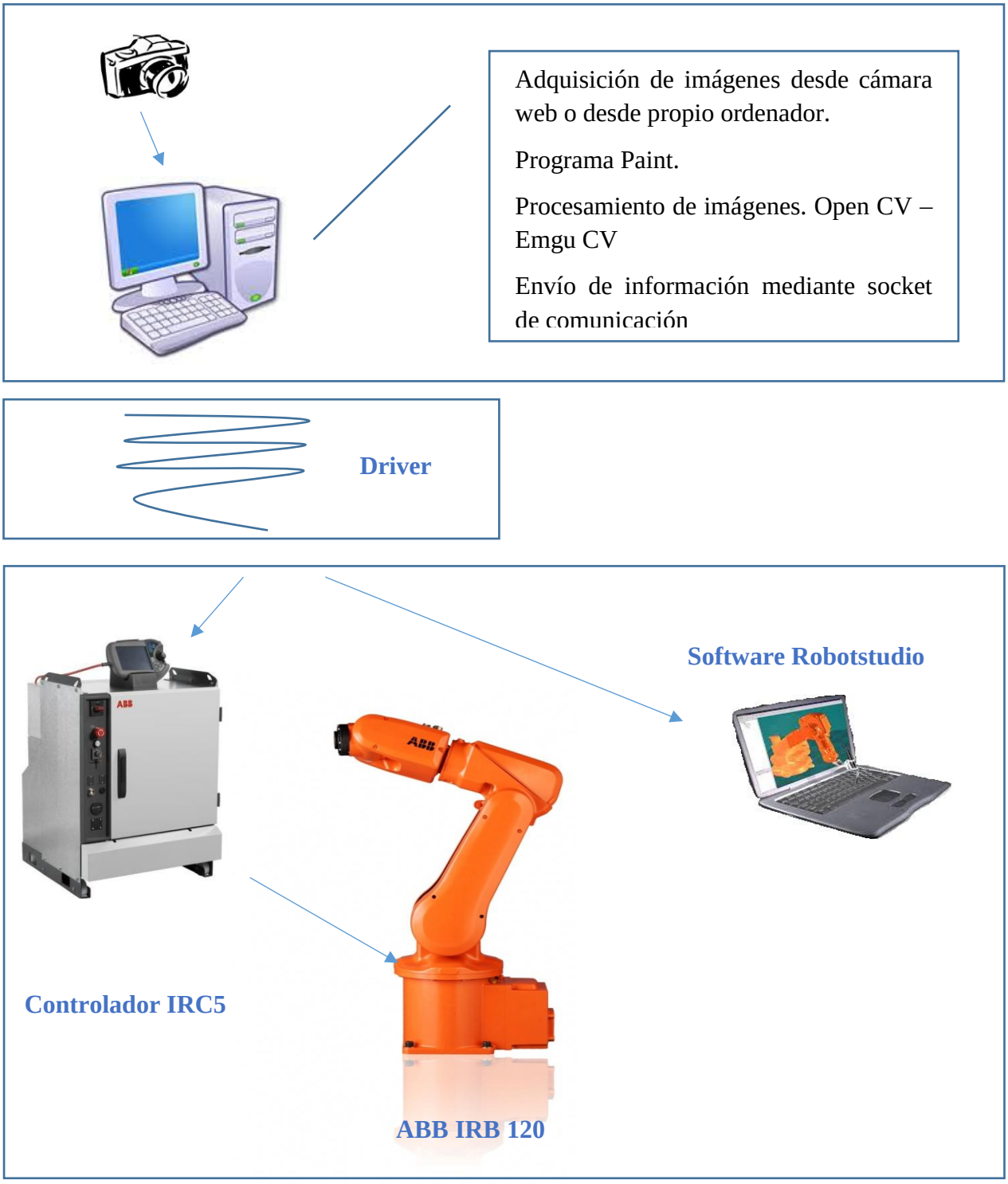


Figura 3.1 Arquitectura general de la aplicación

3.2 Introducción

En los capítulos venideros se comentarán los distintos elementos que conforman el esquema orientativo presentado en el apartado anterior. Si bien no se entrará en profundidad en ellos, se analizarán las funciones de cada uno.

Como se ha mencionado anteriormente, el bloque que atañe principalmente a esta aplicación es el primero, y por lo tanto es en el que más se va a centrar este análisis. Los bloques dos y tres están fundamentados en otro proyecto y no tiene sentido entrar en detalle.

Se puede ver, en el esquema anterior, que el primer bloque está formado por una cámara web (hardware) y un ordenador en el que están instalados los elementos software citados en el capítulo de herramientas utilizadas.

3.3 Adquisición de imágenes

Como el objetivo principal de este proyecto es elaborar una aplicación capaz de realizar un procesamiento de imágenes óptimo, es fundamental adquirir una imagen para que esta pueda ser procesada, si no la aplicación carecería de sentido.

Por lo tanto se fija como punto de partida de la aplicación este punto. La adquisición de una imagen, y como ya se ha mencionado en apartados anteriores, la adquisición de dicha imagen, se podrá realizar mediante tres fuentes principales:

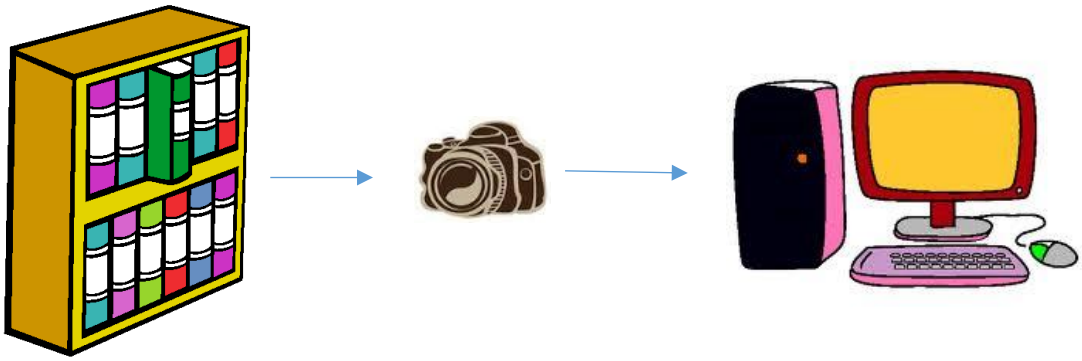
- Adquisición mediante cámara web
- Adquisición desde el propio equipo
- Adquisición mediante programa Paint

3.3.1 Adquisición mediante cámara web

Desde un punto de vista de programación, este es el punto más elaborado a la hora de adquirir la imagen. Pero esta programación no será muy dificultosa debido a la posibilidad de utilizar las bibliotecas de Open CV de la forma especificada anteriormente.

A estas imágenes se le podrán aplicar unos filtros que se detallarán en capítulos posteriores, en apartados pertenecientes al procesamiento de la imagen.

En primera instancia, la aplicación de captura de imágenes respondería a un esquema como el que se muestra a continuación.



Resultado

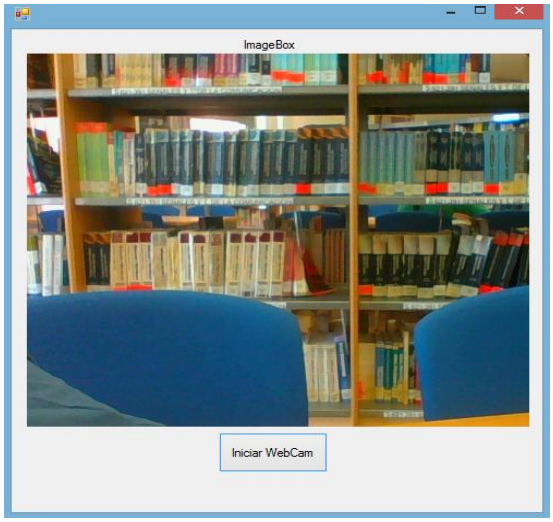


Figura 3.2 Arquitectura de adquisición desde cámara web

3.3.2 Adquisición desde el propio ordenador

La siguiente opción posible, a la hora de adquirir una imagen para la aplicación, es adquirirla directamente desde el propio ordenador. Se programa un buscador de imágenes de los tipos `jpg`, `png`, `bmp` y `tiff`, de forma que se podrá seleccionar la imagen que se desee procesar si está contenida dentro del propio equipo.

Incluir esta funcionalidad en el proyecto resulta muy ventajoso para la aplicación final. El hecho de poder disponer de una imagen para distintas pruebas, permite la posibilidad de una calibración del propio sistema, pues al tratarse de la misma imagen, se podrían eliminar errores sistemáticos. Pero si no se tuviera incluida esta funcionalidad, se harían pruebas con imágenes distintas y características distintas, por lo que los errores en la

prueba de una imagen con los de la siguiente prueba serían distintos y muy difíciles de detectar.

Otras ventajas de esta funcionalidad son que, además de no tener que conectar una cámara web al sistema para que la aplicación funcione, se pueden seleccionar fotos tomadas con cámaras externas en otro tipo de ambientes, si se guardan en el equipo. Por lo que se puede realizar un procesado de cualquier imagen tomada en cualquier parte.

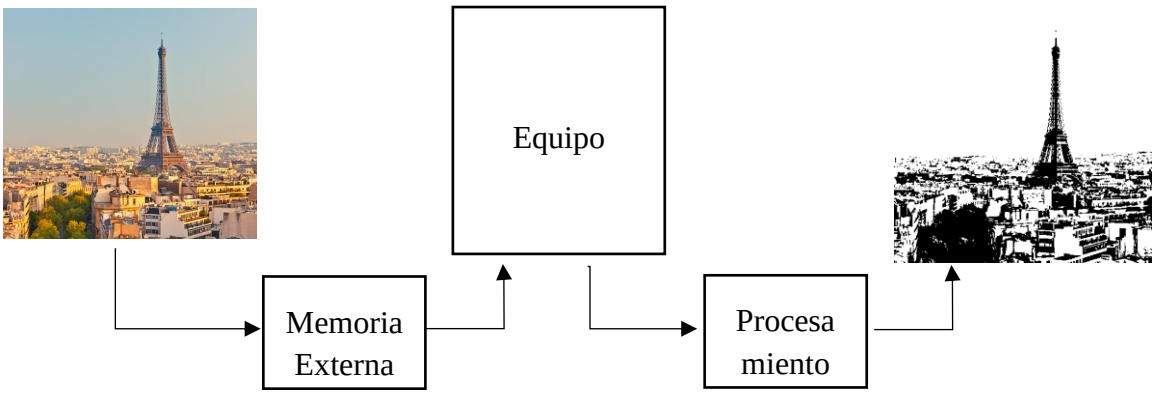


Figura 3.3 Arquitectura de adquisición de imagen importada desde el equipo

3.3.3 Adquisición desde programa Paint

La última fuente de adquisición de la imagen, antes de procesarla, es desde el programa Paint. Se realiza un programa Paint, basado en una aplicación de dibujo capaz de dibujar un trazo que determina la posición del ratón dentro del contenedor correspondiente. Para el uso de esta parte del programa, pulsando el botón izquierdo del ratón se recorre de forma totalmente libre el panel.

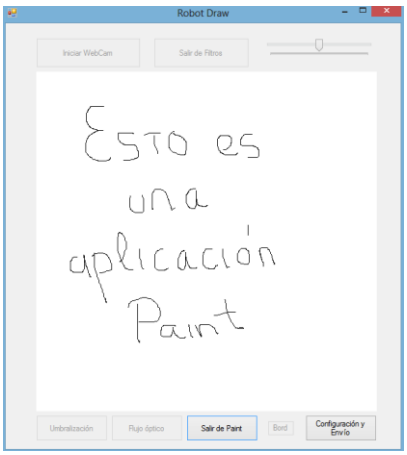


Figura 3.4 Adquisición dibujo Paint

A continuación se propone un esquema orientativo del funcionamiento de las fuentes de adquisición disponibles desde el propio ordenador. Estas son, el programa Paint y todas aquellas imágenes que estén contenidas en el equipo.

Como se aprecia en el esquema, la información no se extrae del exterior, como si ocurría con el caso de la cámara web, si no que toda la información va a estar contenida dentro del propio sistema.

También hay que mencionar que, el socket de comunicación tiene una parte interna dentro del sistema, pero también una parte externa. Por esto en los capítulos iniciales se menciona que había escrita una parte en lenguaje C# y otra en lenguaje RAPID.

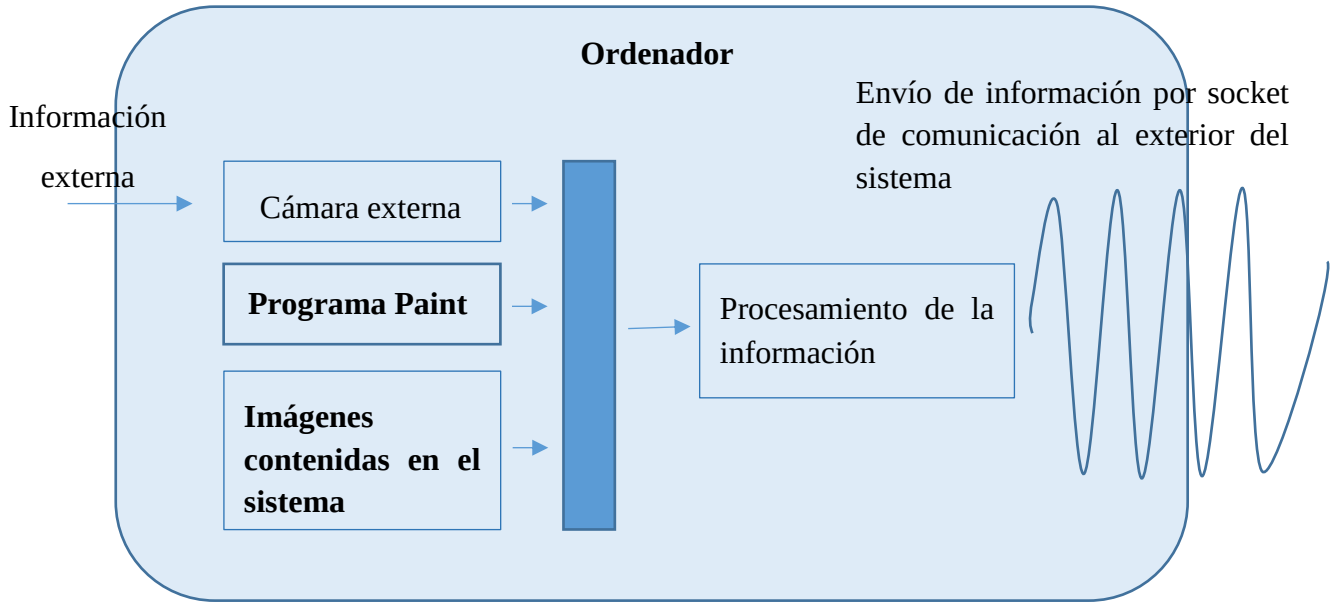


Figura 3.5 Arquitectura de adquisición general y procesamiento

4 Procesamiento de la imagen

En este capítulo se detallarán todos los tipos de filtros disponibles en la aplicación y su funcionamiento. Además se explicará la forma de almacenamiento de las imágenes como información para su posterior envío al brazo robótico.

Para las imágenes adquiridas desde la cámara web se podrán aplicar dos tipos de filtros, un filtro de umbralizado y otro de flujo óptico, mientras que las imágenes adquiridas desde el propio sistema sólo se podrán someter al filtro de umbralización.

4.1 Umbralización

4.1.1 Introducción

La umbralización es uno de los métodos más importantes de segmentación. Su objetivo es convertir una imagen en escala de grises a una nueva con solo dos niveles, de manera que se pueda distinguir entre un cuerpo y un fondo.

Como en todos los métodos de segmentación se trata de asignar cada pixel de la imagen a un grupo (segmento). La imagen a segmentar está compuesta por valores numéricos (uno o más valores de color para cada pixel). Que un pixel pertenezca a un cierto segmento se decide mediante la comparación de su nivel de gris con un cierto valor umbral.

El valor de gris de un pixel corresponde a su nivel de luminosidad y el resto de la información sobre el color no se tiene en cuenta.

Comúnmente los métodos del valor umbral realizan una binarización de la imagen de partida, es decir, se construyen dos segmentos: los objetos y el fondo de la imagen.

La asignación de un pixel a uno de los dos segmentos se consigue comparando su nivel de gris con un valor "t" (threshold) y sigue la siguiente expresión:

$$T_{global}(g) = \begin{cases} 0 & \text{si } g < t \\ 1 & \text{si } g \geq t \end{cases}$$

Figura 4.1 Funcionamiento Umbral

4.1.2 Proceso de umbralización

El objetivo que se persigue con este filtro es transformar una imagen en color, en una matriz bidimensional de unos y ceros en los que reside la información de dicha imagen.

Pero para lograr esto es necesario realizar un procesado intermedio. Se requiere por encima de todo, una imagen. Las fuentes de adquisición de una imagen ya se han detallado en capítulos anteriores, y con ellas se obtiene una imagen en color (RGB).

A esta imagen RGB se le aplican una serie de funciones que se pueden encontrar entre las contenidas en Open CV, obteniendo así una imagen en escala de grises.

Esto es, una imagen con 256 niveles de gris, de los cuales tiene que quedar claro que el valor 0 es el negro y el valor 255 es el blanco, y entre ellos una escala de grises.

La imagen siguiente demuestra el proceso sufrido por la imagen de la izquierda, una imagen en RGB, y la de la derecha, la propia imagen RGB tras ser convertida a escala de grises.

Pero el objetivo del proyecto era pintar imágenes binarias, y para eso se debe acotar estos 256 niveles de gris de los que se habla, a únicamente dos. Para ello se aplica otra función, esta vez para incluir un umbral, con la que se dividen estos 256 valores en dos segmentos de 128 valores.

De cara a la aplicación actual se fijará el valor del umbral en 127, esto quiere decir que todos aquellos valores cuyo valor de luminosidad sea inferior a 127 se igualarán al valor 0 (negro) y todos aquellos que sean iguales o superiores a dicho umbral se igualarán a 255 (blanco).

Con la intención de que sea una aplicación algo más interactiva se incluye un cursor, que se detallará en capítulos posteriores, con el que variar el valor del mismo umbral, permitiendo así al usuario experimentar con la imagen y visualizar los cambios que se suceden.

En este proceso de umbralización, de manera esquemática, se pueden distinguir tres partes principales:

- Inicio
- Cuerpo
- Fin

Es importante separar estas tres partes, ya que algunos son procesos ocultos que no son visualizados por pantalla y, por lo tanto, pasan desapercibidos para el usuario que ejecute la aplicación, pero que sin duda merece la pena mencionar.

Las partes Inicio y Fin, además de llevar implícita una programación interna en la aplicación, estas se muestran por pantalla de tal forma que son las únicas referencias que tiene el usuario del estado en el que se encuentra la imagen que está procesando.

Sin embargo, el Cuerpo del programa es invisible para el usuario.

A continuación se indica mediante un esquema este proceso, indicando cada una de las partes principales.

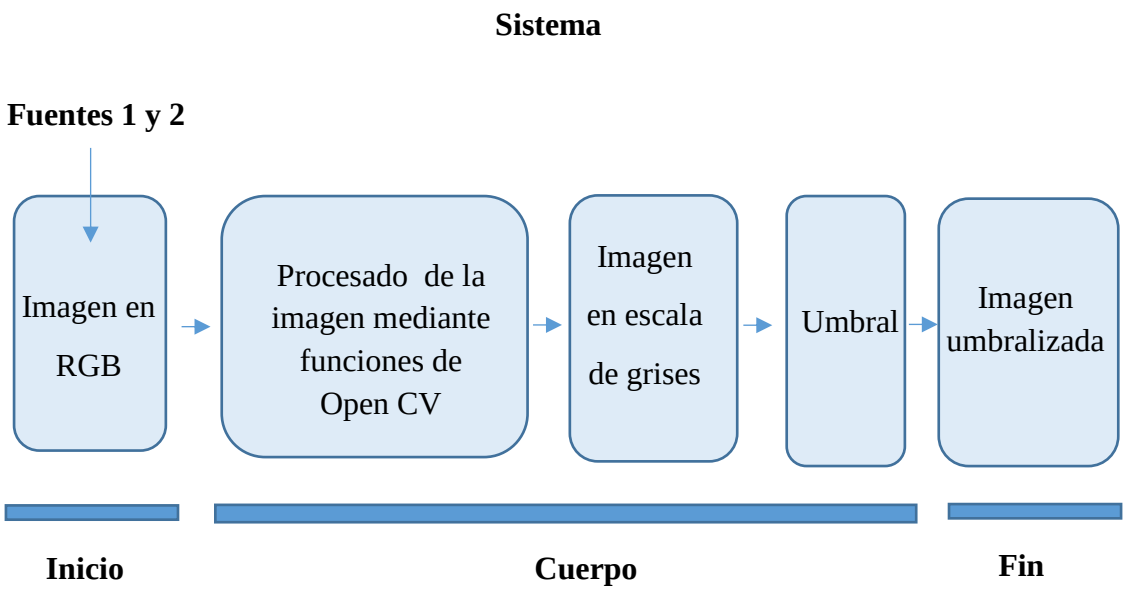


Figura 4.2 Procesamiento de imágenes. RGB a niveles de gris

Como se ha mencionado en apartados anteriores, el filtro de umbralización está disponible para aquellas imágenes obtenidas directamente desde la cámara web y aquellas que se adquieren directamente del propio sistema, estas han sido llamadas fuente 1 y fuente 2 respectivamente en el esquema.

Para el resto de funcionalidades como puede ser la aplicación Paint o el filtro de flujo óptico no tiene ningún sentido incluir este proceso.

A continuación se presentarán algunas imágenes que demuestran lo contado durante este capítulo.

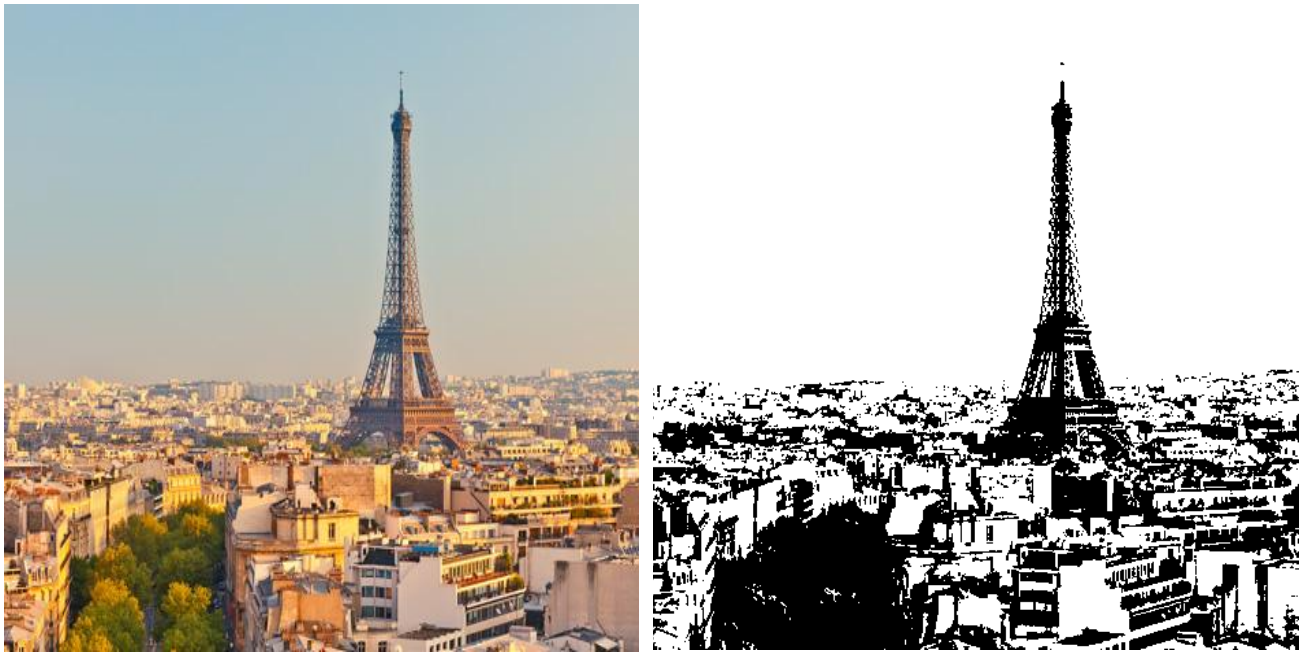


Figura 4.3 Imagen importada desde el ordenador y su umbralización

Figura 4.4 Imágenes sometidas al proceso de umbralización

4.2 Flujo óptico

4.2.1 Introducción

Se conoce el flujo óptico como el patrón del movimiento aparente de los cuerpos, superficies y bordes en una escena causado por el movimiento relativo entre el observador y la escena.

El concepto surgió por primera vez en la década de los cuarenta, y finalmente fue publicado por James. J. Gibson.

El flujo óptico encuentra multitud de aplicaciones en la sociedad actual tales como la detección de movimiento, el tiempo de colisión, la codificación del movimiento compensado o la segmentación de objetos, entre muchas otras.

El cálculo del flujo óptico no es sencillo y computarlo para tomas realizadas en un ambiente real puede llegar a ser extremadamente difícil. Por eso los algoritmos actuales se basan en determinadas hipótesis que generalmente no se cumplen en escenarios reales.

Merece la pena resaltar la importancia de los vectores de movimiento, estos vectores son una estimación del desplazamiento horizontal y vertical de cada región de una cierta imagen con respecto a varios frames de la secuencia.

Figura 4.5 Ejemplo funcionamiento vectores de posición

Esta imagen representa a la perfección el concepto explicado anteriormente, se pueden ver los vectores que indican el movimiento de cada pixel.

En cualquier caso, la funcionalidad que compete a esta aplicación no será bajo ningún concepto una desarrollo matemático – científico de la teoría del flujo óptico. Simplemente se realizará una sencilla estimación con la cual aproximar dicho efecto.

4.2.2 Proceso de flujo óptico

El objetivo de este filtro no es otro que transformar una imagen en color, en una imagen en la que solo figuren los vectores de movimiento de los píxeles móviles. Y una vez conseguido esto, almacenar esos vectores como trazos, para que el robot los represente.

Este proceso solo puede tener lugar cuando las imágenes obtenidas se están tomando mediante la cámara web, ya que para que funcione, se necesita comparar dos frames separados un instante de tiempo. Con lo que, la única manera de hacerlo, será mediante una adquisición continua de imágenes desde la cámara conectada al equipo.

El proceso que sigue esta funcionalidad es el siguiente:

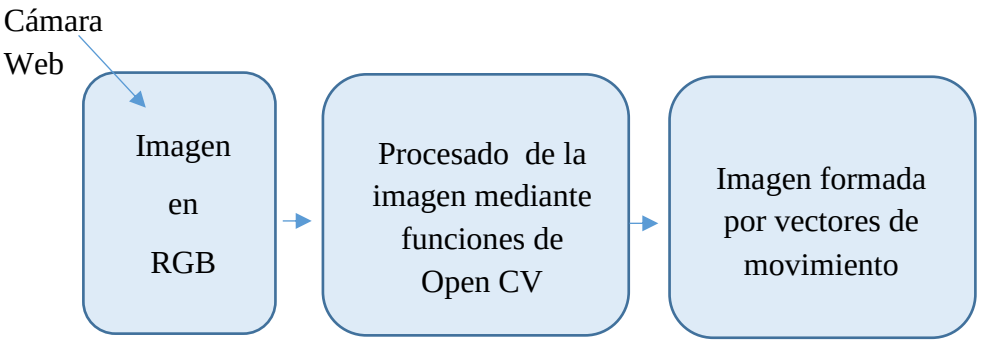


Figura 4.6 Procesamiento imagen. RGB a vectores de posición

Como se muestra en el esquema, la imagen proveniente del exterior del equipo, a través de la cámara web, se inicializa en el contenedor imágenes de tal forma que el usuario está viendo la captura. A continuación, en el bloque de programación se crean una serie de imágenes, con ayuda de Emgu CV, con las cuales se realizan las operaciones necesarias para detectar el movimiento de un pixel. Una vez detectado este movimiento se dibuja un trazo entre punto inicial y punto final, lo que se entiende como el vector de movimiento de dicho pixel.

Tras realizar estas operaciones, por pantalla se visualiza una imagen en blanco en la que, cuando la cámara web detecta movimiento, se trazan los vectores de movimiento que se mencionaban. Lo que le permite al usuario comprobar que el procesamiento ha sido satisfactorio, y le deja potestad de tomar la captura adecuada.

Se presenta una captura realizada con esta funcionalidad por la propia aplicación, como se puede apreciar, los vectores de movimiento dejan entrever la silueta que ha provocado el movimiento de los píxeles.

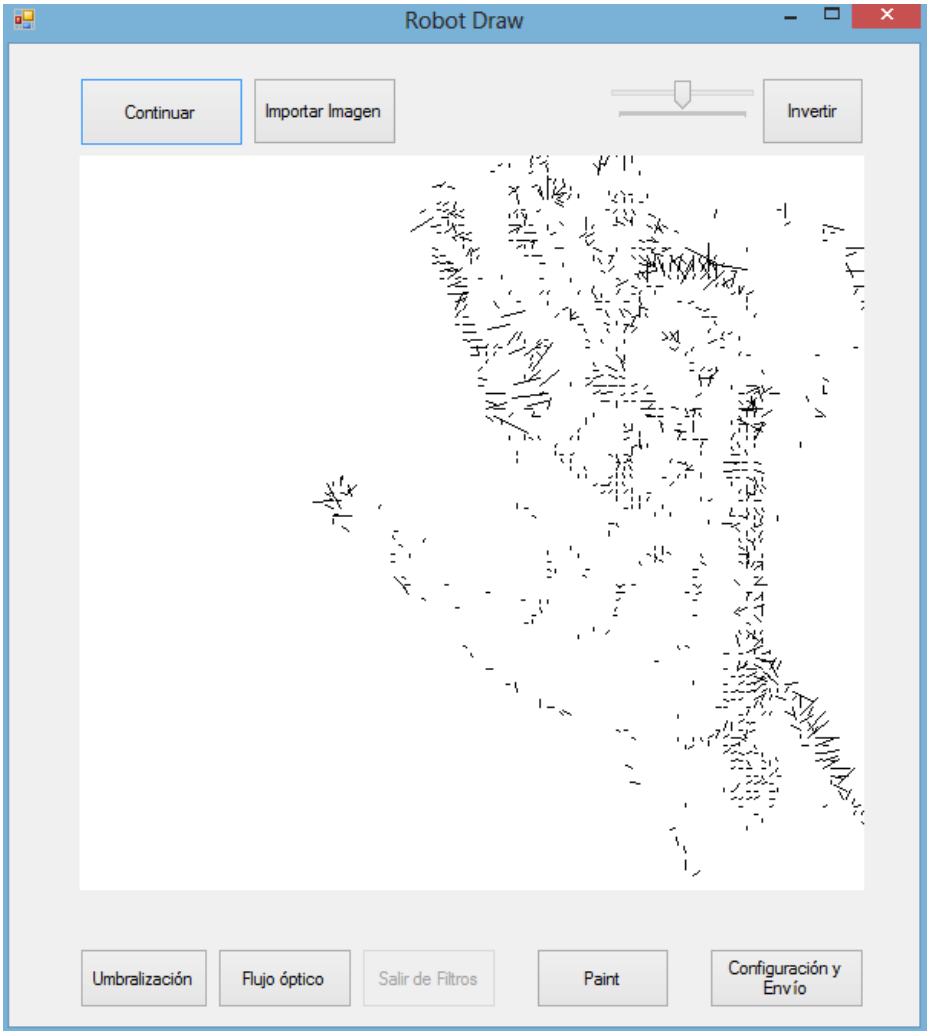


Figura 4.7 Imagen captada con cámara web. Filtro flujo óptico

4.3 Almacenamiento de la información procesada

Una vez se ha procesado la imagen adquirida desde cualquiera de las fuentes y se le ha aplicado el filtro deseado, se obtiene, según cada caso, un tipo de imagen. Ahora el objetivo debe ser poder transformar esa imagen, en información accesible por el sistema, de tal manera que posteriormente se pueda enviar a través del socket de comunicación proporcionado al brazo robótico en forma de coordenadas X Y de un plano.

4.3.1 Almacenamiento del filtro Umbralización

Se recuerda que al final del proceso se obtenía una imagen en dos niveles de gris, con lo que se pueden proponer diferentes soluciones de almacenamiento viables para, posteriormente, tener una correcta representación.

Los dos niveles de gris conforman una imagen en blanco y negro, que se podría equiparar de cierta forma a un mapa de bits donde los píxeles negros fueran unos y los píxeles blancos fueran ceros.

De esta forma, se dispondría de una imagen de tamaño ancho por largo, en píxeles, como si de una matriz bidimensional se tratase. Siendo cada una de las celdas de la matriz un cero o un uno, que hacen referencia a la estructura de la propia imagen.

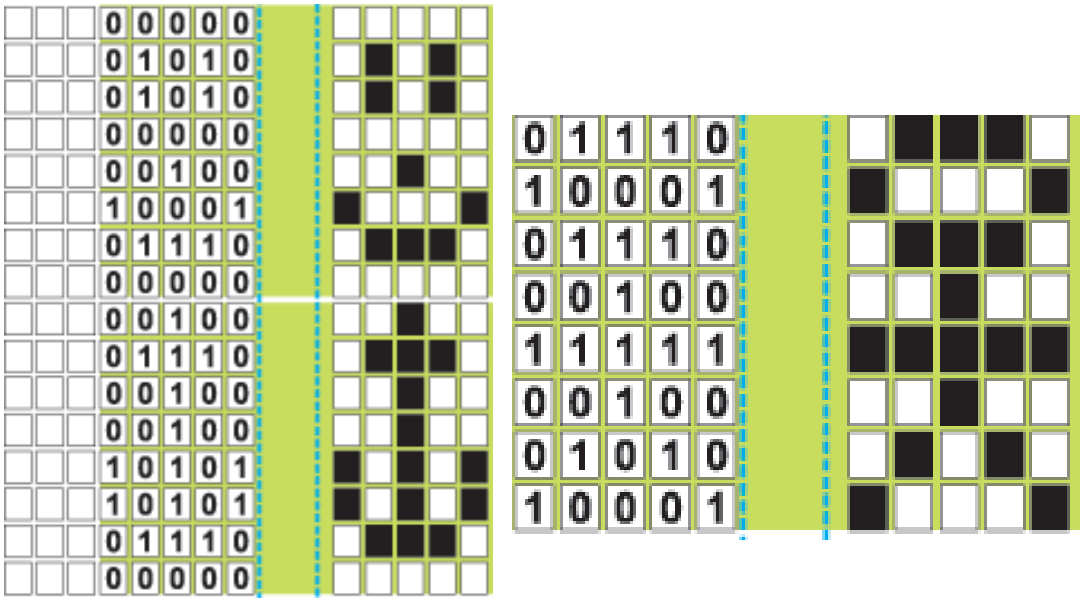


Figura 4.8 Ejemplos mapas de bits

Estas imágenes representan a la perfección lo citado en el párrafo anterior, y de manera similar se comporta la aplicación desarrollada.

Para la aplicación actual se puede imaginar la imagen procesada como una matriz denominada “Matriz_Imagen”, en la cual se muestra la información correspondiente a la imagen procesada.

Por ejemplo, en el caso de la imagen de la derecha, Matriz_Imagen respondería así:

Matriz_Imagen en contenedor ImageBox

0,0 0	0,1 1	0,2 1	0,3 1	0,4 0
1,0 1	1,1 0	1,2 0	1,3 0	1,4 1
2,0 0	2,1 1	2,2 1	2,3 1	2,4 0
3,0 0	3,1 0	0,2 1	3,3 0	3,4 0
4,0 1	4,1 1	4,2 1	4,3 1	4,4 1
5,0 0	5,1 0	5,2 1	5,3 0	5,4 0
6,0 0	6,1 1	6,2 0	6,3 1	6,4 0
7,0 1	7,1 0	7,2 0	7,3 0	7,4 1

Tabla 4.1 Posicionamiento y discriminación en mapa de bits

Si se equiparase “Matriz_imagen” a un plano en el que se ha hecho una cuadrícula donde cada celda fuera la posición exacta de la celda en dicha cuadrícula, se tendría una estructura simple en la que resultaría tremendamente sencillo determinar que objeto o valor hay en cada posición.

Entendiendo la estructura interna de la imagen procesada con el ejemplo propuesto de la matriz “Matriz_Imagen”, ahora se requiere almacenar la información “útil” de dicha estructura.

Esta información se debe almacenar en otra matriz, esta vez real, en la que se almacene la posición exacta de todos aquellos píxeles negros de la imagen. Para cumplir tal propósito se implementa un bucle que recorre la matriz “Matriz_Imagen”, evaluando celda a celda el valor del dato contenido.

Por lo tanto ya se conoce la forma de evaluar cada celda, cada pixel de la imagen de dos niveles de gris y tener acceso a la posición de aquellos píxeles que representen un tono negro en la imagen.

A continuación, estas posiciones “útiles” deben ser almacenadas en otra matriz, esta vez real, que se denominará “Matriz_Lineas”. Pero esta matriz no contendrá valores, contendrá posiciones.

Además en cada celda de esta matriz se dará soporte para que pueda almacenar hasta dos posiciones y no sólo una. Esta característica se explicará en las próximas líneas.

Conceptualmente sería algo como lo que se presenta en el la siguiente tabla, donde como se ha especificado cada celda de la matriz tiene capacidad para hasta dos posiciones, con coordenadas X Y cada una de ellas.

MATRIZ LÍNEAS

P0 = (Xi, Yj) P1 = (Xi, Yj)	P0 = (Xi, Yj) P1 = (Xi, Yj)	P0 = (Xi, Yj) P1 = (Xi, Yj)	P0 = (Xi, Yj) P1 = (Xi, Yj)
P0 = (Xi, Yj) P1 = (Xi, Yj)	P0 = (Xi, Yj) P1 = (Xi, Yj)	P0 = (Xi, Yj) P1 = (Xi, Yj)	P0 = (Xi, Yj) P1 = (Xi, Yj)
P0 = (Xi, Yj) P1 = (Xi, Yj)	P0 = (Xi, Yj) P1 = (Xi, Yj)	P0 = (Xi, Yj) P1 = (Xi, Yj)	P0 = (Xi, Yj) P1 = (Xi, Yj)
P0 = (Xi, Yj) P1 = (Xi, Yj)	P0 = (Xi, Yj) P1 = (Xi, Yj)	P0 = (Xi, Yj) P1 = (Xi, Yj)	P0 = (Xi, Yj) P1 = (Xi, Yj)

Tabla 4.2 Almacenamiento teórico de posiciones

Con esta excepcionalidad, se consigue un método válido para la implementación de las dos posibles opciones de almacenamiento de la información de las que se habla al comienzo del capítulo:

- Almacenamiento de puntos
- Almacenamiento de líneas

4.3.1.1 Almacenamiento de puntos

Parece que el modelo propuesto está pensado únicamente para el almacenamiento de líneas y que posteriormente el robot solo va a ser capaz de representar trazos y no puntos aislados, pero no es así.

Se realiza la programación con el fin de que cada celda pueda almacenar hasta dos posiciones, como se ha mencionado, pero existe la posibilidad de almacenar una única posición, con lo que además de trazos también se pueden almacenar puntos aislados.

Con lo cual, para el caso actual en el que sólo se estén almacenando puntos, la matriz Líneas quedará de la siguiente manera:

	C1	C2	C3	C4	C5	C6	C7	C8
Fila 0	0	1	0	1	1	1	0	1

Tabla 4.3 Almacenamiento teórico de puntos en filas de la matriz

Matriz_Líneas

P0 = (0, 1)	P0 = (0, 4)	P0 = (0, 5)	P0 = (0,6)	P0 = (0,7)

Tabla 4.4 Almacenamiento práctico de puntos en filas de la matriz

Esta Matriz_Líneas, es el contenedor de información del que dispone el sistema y con el cual el robot podrá representar con precisión la imagen procesada.

4.3.1.2 Almacenamiento de líneas

En este caso se completan las dos posiciones que admite cada celda, de tal forma que estas dos posiciones van a conforman una línea.

Esto quiere decir, que el sistema comprenderá que cada celda contiene una posición inicial y una posición final a la que el robot tiene que ir, sin levantar la herramienta de la posición programada para que pinte en el plano.

El objetivo de este método es identificar píxeles consecutivos en cada línea de la matriz, esto se lleva a cabo mediante un rastreo secuencial, punto a punto de todas y cada una de las líneas de la matriz. Y en función del valor contendido en cada uno de ellos, se almacena la información o se sigue muestreando.

Se presenta un esquema para entender mejor este funcionamiento.

	C1	C2	C3	C4	C5	C6	C7	C8
Fila 0	0	1	0	1	1	1	0	1

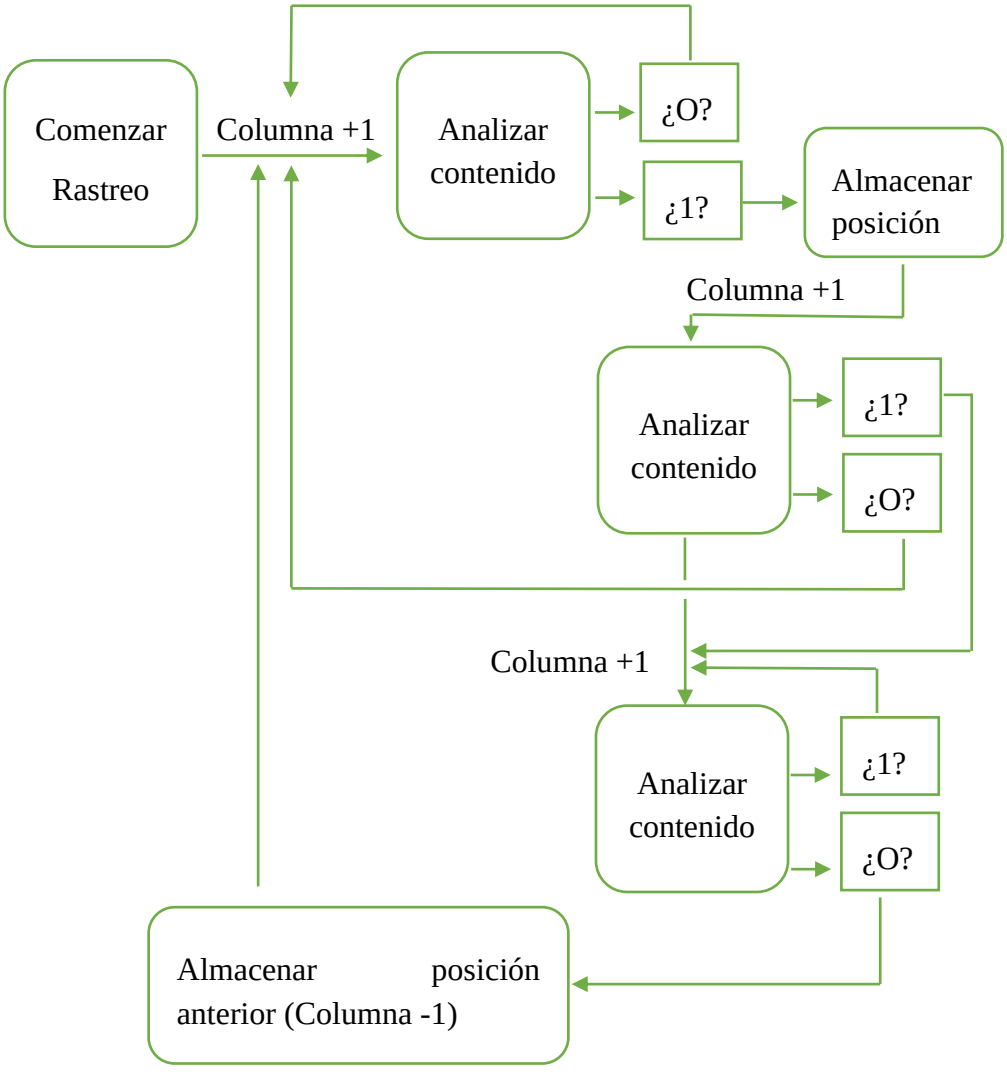


Figura 4.9 Equema de funcionamiento de rastreo en umbralización

Explicado de otra manera, se recorre cada fila de la matriz Matriz_Imagen en busca de celdas con un valor de gris equivalente al color negro, y según el valor de cada celda se almacena una posición o no se almacena.

En el ejemplo mostrado, se inicia el rastreo en busca de unos, por lo que se evalúa cada celda de la fila. Si esta contiene un 0 se pasa a la siguiente y si contiene un 1 se almacena la posición de la celda que lo contiene.

Además, si la celda siguiente a la celda que contenía un 1, también contiene un 1, entonces se trata de píxeles consecutivos por lo que no se almacena la posición de la última celda y se sigue recorriendo la fila hasta encontrar una celda que contenga un 0, valor que expresa el fin de los píxeles consecutivos y por lo tanto el fin de una línea de puntos. Se almacena la posición de la celda previa a la que contiene este último 0.

De esta forma, para el ejemplo anterior, se almacenaría la información en Matriz_Lineas de la siguiente forma.

	C1	C2	C3	C4	C5	C6	C7	C8
Fila 0	0	1	0	1	1	1	0	1

Matriz_Líneas

P1 = (0, 2)	P1 = (0, 4) P2 = (0, 6)	P1 = (0, 8)

Tabla 4.5 Almacenamiento de los puntos que conforman rectas

Esta Matriz_Líneas es la que se utiliza para enviar la información al robot, con lo que su correcta construcción es muy importante. El método de envío de esta información se detallará en capítulos posteriores de manera superficial, ya que para esto se ha utilizado el socket de comunicación proporcionado antes de comenzar la práctica.

Aunque se debe aclarar que para el correcto funcionamiento del método de envío se ha debido respetar el protocolo de comunicación del socket, trabajando sobre el mismo.

4.3.2 Almacenamiento del filtro Flujo óptico

Se recuerda que tras el proceso llevado a cabo por el flujo óptico se obtenía una imagen blanca en la que solamente figuraban aquellos vectores provocados por el movimiento de determinados píxeles.

En esta ocasión no tiene sentido implementar un algoritmo que recorra la imagen en busca de píxeles negros para conocer su posición, dado que no resultaría atractivo hacer una representación por puntos o por líneas horizontales como las planteadas en el capítulo anterior.

Este método comparte con el anterior, la funcionalidad de la matriz Matriz_Líneas, en la cual se podrá almacenar hasta dos posiciones por cada celda de la matriz.

Además también son de gran importancia las funciones de la librería de Open CV ya que son estas las que devuelven los argumentos necesarios para realizar este análisis.

El propio sistema identifica que píxel se ha movido respecto a un frame anterior, y este traza un vector o segmento de origen y fin, la posición previa y fin del pixel.

De esta forma, se consigue el punto inicial y el punto final del vector y se almacena en la matriz Matriz_Lineas. Por tanto, esta matriz contendrá en cada una de sus celdas dos posiciones que conforman un segmento. Independientemente de que sean horizontales o no lo sean, el robot entenderá que entre estos dos puntos no debe modificar la altura de su herramienta, trazando una línea entre ellos.

Esquema:

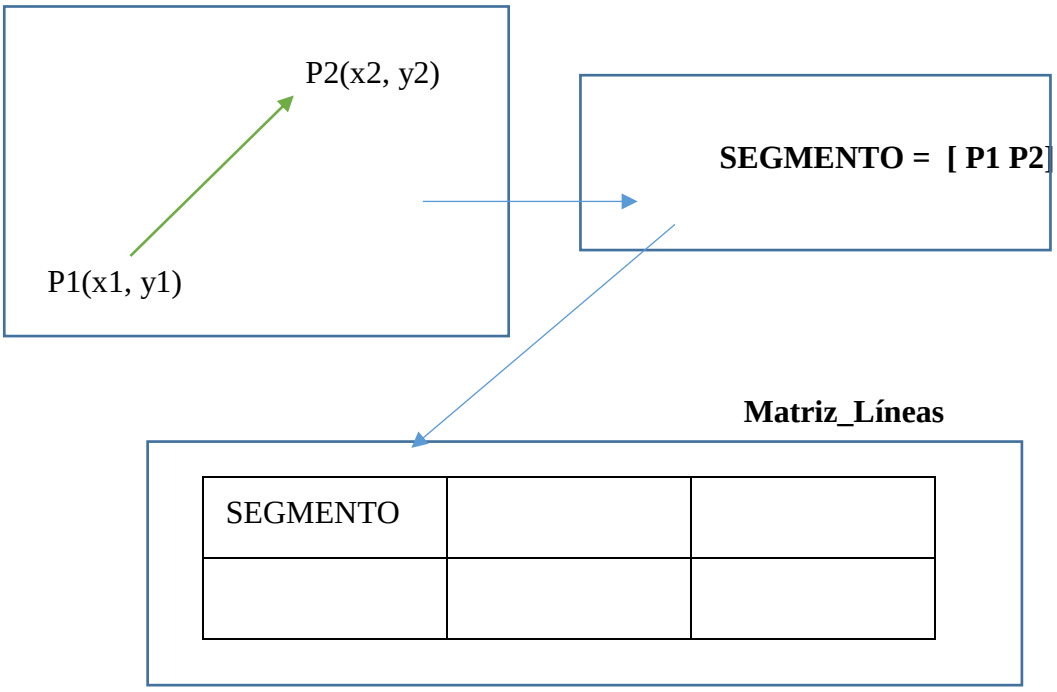


Figura 4.10 Almacenamiento teórico del vector de posición

La idea se constituye de tres pasos: la creación del vector, la identificación puntos del segmento y por último, añadir este segmento a la matriz en cuestión.

Por consiguiente, en un caso más complicado como es el de la aplicación simplemente se almacenarán más segmentos, cada uno de ellos en las celdas de la matriz.

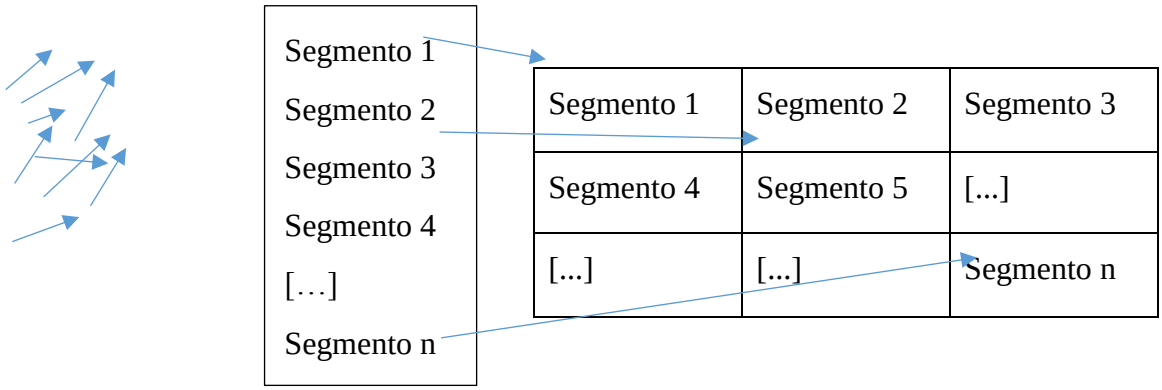


Figura 4.11 Almacenamiento práctico de la imagen como vectores de posición

De esta manera se organizan los segmentos identificados mediante la propia función de Open CV.

Lo único que hay que añadir es que se incluyen unos mínimos para los que, si un píxel no se ha movido más de una cierta distancia, este no se considera. En el caso de que supere la distancia mínima, este se considera y se almacena en la matriz de almacenamiento de datos.

5 Paint

Se incluye como parte del programa una funcionalidad Paint que simule un programa de dibujo. Esta funcionalidad aporta al conjunto una posibilidad más que hace que la aplicación final sea mucho más completa. No solo se puede adquirir imágenes, si no que con esta característica también se podrán realizar trazos libres de dibujo que el robot representará en el plano como si fuera el mismo usuario con un rotulador, salvo que con un problema computacional como paso intermedio.

5.1 Funcionalidad Paint

En el mismo contenedor de imágenes que se había utilizado para los tipos de adquisición anteriores, ahora se utiliza como si de una pizarra – panel se tratase.

Se programa la aplicación para que, cuando el usuario desplace el ratón por encima de esta pizarra, se dibuje una línea entre todos los puntos por los que pase el ratón. De esta manera se obtiene el efecto de dibujo de una polilínea entre puntos adyacentes.

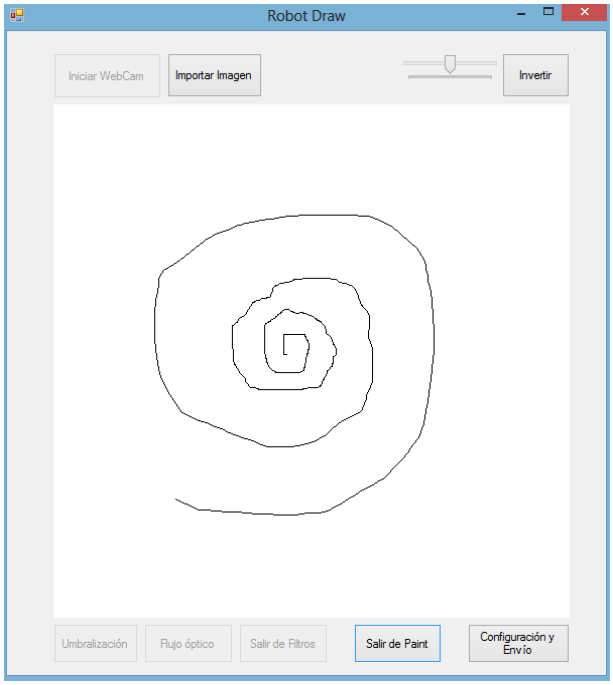


Figura 5.1 Ejemplo realización dibujo Paint

5.2 Almacenamiento de los datos

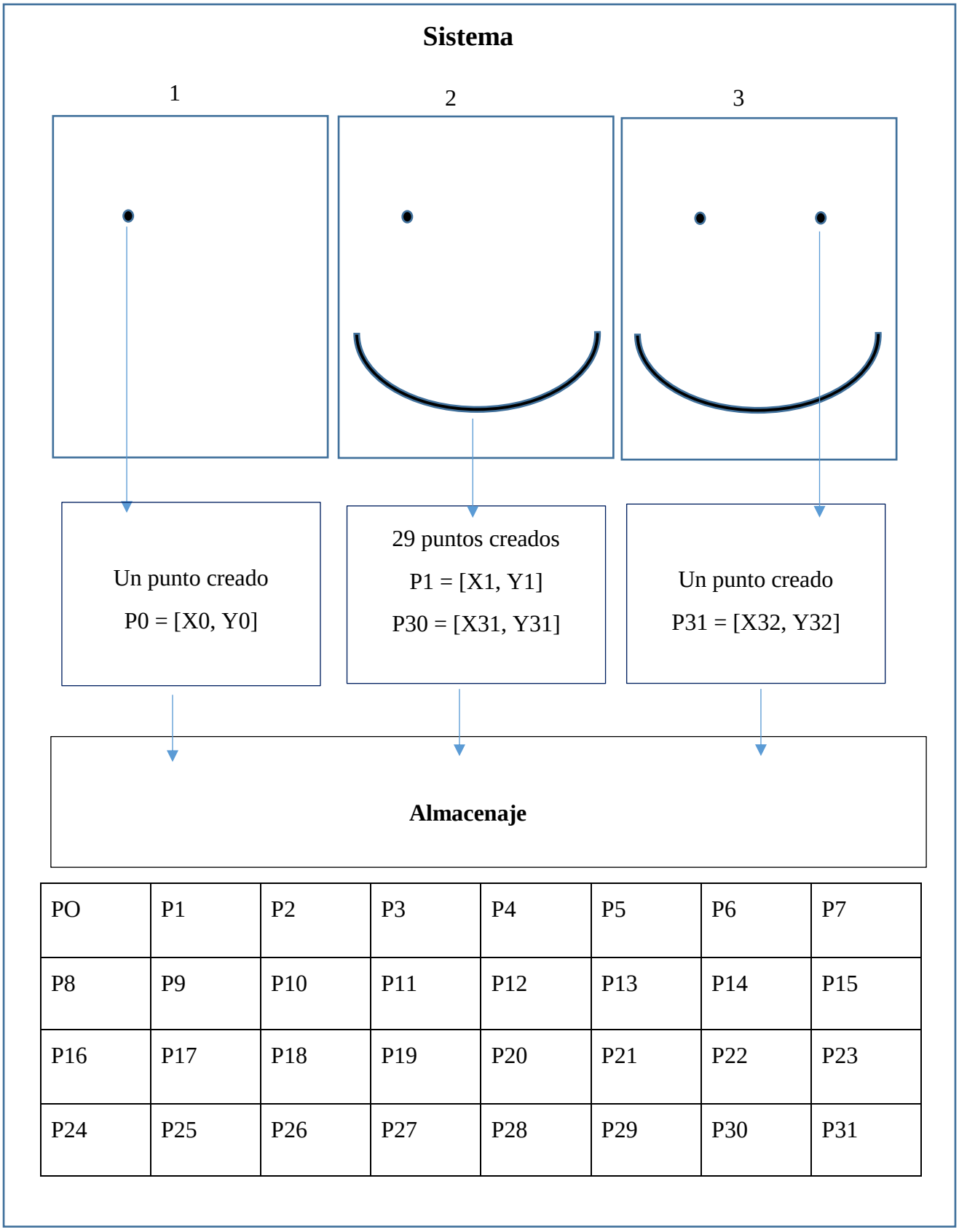
En este caso en concreto se utiliza una lógica de almacenamiento similar a la de la información de los tipos de adquisición anteriores. Esto es, que el panel virtual donde a priori se va a realizar el dibujo, como los contenedores de los capítulos anteriores, son planos y por lo tanto tienen dos dimensiones, una dimensión X y una dimensión Y.

Los puntos que vaya determinando el usuario por los cuales se traza el dibujo son almacenados en una matriz del tipo especificado en los capítulos anteriores, esta vez guardando en cada celda una única posición, cada punto que representa la polilínea, y no dos puntos como en algún caso anterior.

Esta vez no se hará un almacenaje secuencial en cuanto al origen de coordenadas, también ocurría esto con el flujo óptico pero no con el filtro umbralización, si no que esta vez será el propio usuario quien determine el orden de este almacenamiento según en trazado que realice.

Dicho de otra manera, los puntos se almacenan en función de su creación, de forma cronológica. Utilizando esta forma de almacenaje el robot, cuando reciba los datos, pintará el dibujo tal y como lo ha pintado el usuario, siguiendo el mismo orden.

En la página siguiente se detalla un ejemplo práctico que facilitará la comprensión de esta explicación.



PO	P1	P2	P3	P4	P5	P6	P7
P8	P9	P10	P11	P12	P13	P14	P15
P16	P17	P18	P19	P20	P21	P22	P23
P24	P25	P26	P27	P28	P29	P30	P31

Figura 5.2 Esquema de almacenamiento de puntos Paint

6 Envío de la información

6.1 Introducción al Socket de comunicación

Una vez se han adquirido las imágenes desde las fuentes disponibles como la cámara web, el propio equipo o la aplicación Paint, se ha realizado un procesamiento de la imagen en aquellos casos habilitados para ello y se ha llevado a cabo un correcto almacenamiento de la información procesada, es momento de plantearse qué hacer con toda esta información almacenada para que el robot pueda trabajar. Es aquí donde entra en juego el socket de comunicación que se ha proporcionado antes del comienzo de este proyecto, y en el cual se ha basado esta aplicación para la comunicación con la estación del robot.

Las especificaciones técnicas del mismo se pueden encontrar de forma concisa en la memoria del proyecto "Socket based communication in RobotStudio for controlling ABB-IRB120 robot. Design and development of a palletizing station" de Marek Frydrysiak.

6.2 Uso de las matrizes de almacenaje

Como se ha visto durante los capítulos previos a este, se han tratado temas de almacenaje en los que, para construir esta aplicación, se han utilizado matrices auxiliares para facilitar el trato de la información. En ellas, se han almacenado las posiciones "del contenedor – pizarra" que se quieren utilizar para que el robot las implemente.

Se dice que se han guardado las posiciones del "contenedor – pizarra" porque, como se ha mencionado anteriormente, se planteó que el contenedor ImageBox se trataría como si de un plano se tratase con sus coordenadas X Y correspondientes, en los que se reflejarían sólo los puntos cuyo valor de gris fuera uno determinado para el caso de la umbralización, o puntos concretos en los casos del flujo óptico y la aplicación Paint.

Pero esta información no puede aplicarse de forma directa sobre el socket de comunicación para que sea trasladada al robot, ya que existe una diferencia notoria entre las dimensiones del ImageBox y las del plano en el que se va a realizar la representación de la imagen. La información que tenemos es de la posición del píxel respecto al origen de coordenadas del contenedor, y en el plano de representación del robot se debe cotejar como la posición, en milímetros respecto a un origen fijado por el usuario.

Por lo tanto, justo antes de realizar el envío de la información se debe aplicar una función capaz de realizar una transformación de estos puntos a un "formato" acorde al plano donde se va a realizar esta representación. Sigue un proceso como el mostrado en el esquema:

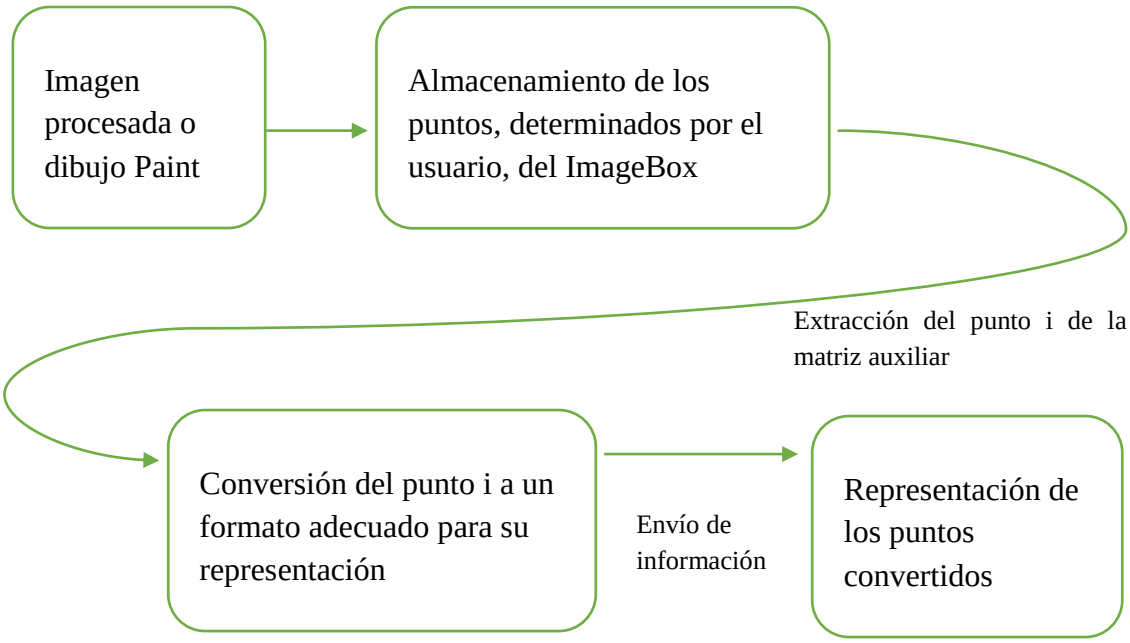


Figura 6.1 Uso de la informacion procesada y envío al robot

El algoritmo implementado para esta conversión se presentará con más detalle en el capítulo de planos, pero de una manera superficial se puede explicar con el siguiente esquema:

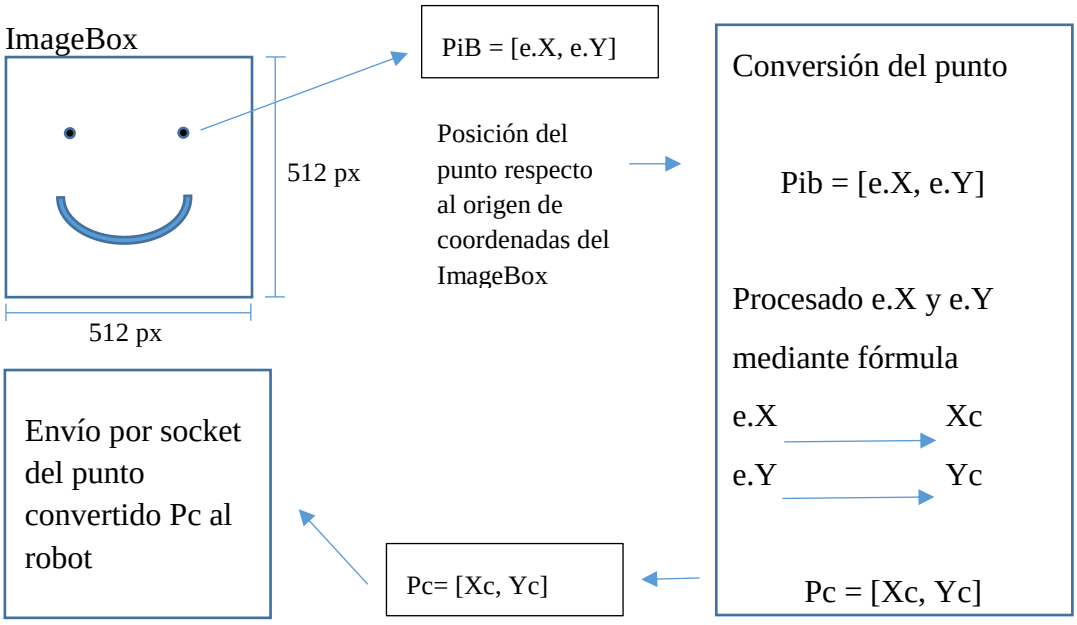


Figura 6.2 Algoritmo para la conversión dimensión ImageBox - dimensiones del plano

Una vez se tiene definida la fórmula para realizar la conversión del punto y se poseen las matrices de datos donde se ha almacenado toda la información correspondiente a las imágenes lo único que hay que hacer es construir un bucle que recorra las matrices, celda a celda, aplicándoles la fórmula de conversión antes de lanzar el dato al sistema de comunicación.

6.3 Uso de los datagramas

Como se ha mencionado, cada punto útil es sometido a un proceso de transformación para que se adapte a las unidades y dimensiones del plano en el que se va a representar la imagen. Cada punto convertido, Pc, tenía una componente X y una componente Y pero no una componente Z, dado que la representación de la imagen se va a llevar a cabo con muy pocos valores distintos de la componente Z.

Así pues, si Pc = [Xc, Yc], se aprovechan las funciones contenidas en el socket de comunicación proporcionado al comienzo del proyecto para enviar la información.

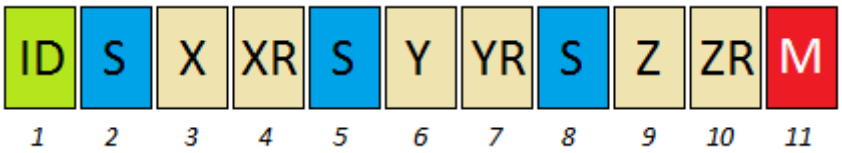
La función en concreto que tiene como objetivo el envío de la posición es:
Sendpoint(); →Y como argumentos, en este caso, se le dan :

- PosValue[0] = p.X;
- PosValue[1] = p.Y;
- PosValue[2] = posZ;

Siendo p.X y p.Y, los valores del punto convertido. En el caso de p.Z su valor se establece de forma manual, fijándose su valor en dos valores concretos:

- Zarriba: posición intermedia entre trazos. Fijada a una altura de 3 x X mm por encima del plano.
- Zabajo: posición del punto o del inicio de cada traza. Fijada a una altura de Xmm por encima del plano (no es 0mm por la herramienta).

De manera más profunda, para el envío de estos puntos se utilizan estructuras denominadas datagramas, que han sido desarrolladas para el socket de comunicación que se ha citado anteriormente. Estas estructuras, para el envío de la posición del TCP del robot, tienen la siguiente forma:



A continuación se presenta un esquema orientativo en el que se puede ver de manera intuitiva los pasos que sigue la aplicación para el envío – representación de la imagen y las funciones básicas correspondientes utilizadas.

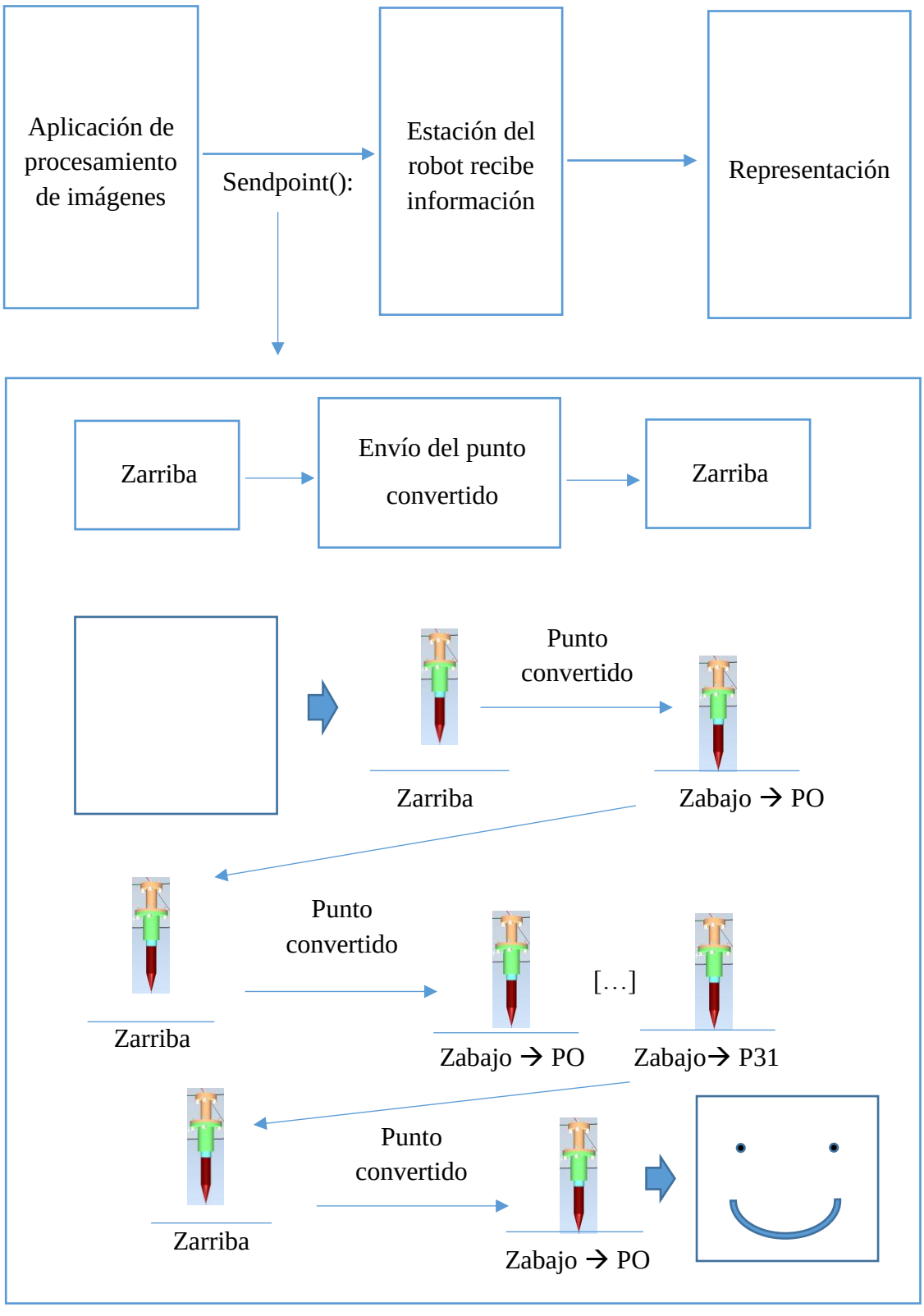


Figura 6.3 Proceso de dibujo. Posiciones y herramienta

6.4 Configuración del envío

Para una aplicación más completa, se han incluido una serie de opciones que la habilitan a ofrecer varios formatos en los que el robot puede realizar la representación de la imagen.

Como la imagen procesada o el dibujo realizado con el Paint se va a plasmar en un plano, se fijan tres formatos en los cuales se puede hacer dicha representación. Estos estándares serán los correspondientes a los tamaños A3, A4 y al tamaño específico de una pizarra.

Figura 6.4 Estándares utilizados para la representación

Además de la posibilidad de cambio de tamaño, también se añade la opción de escoger si la imagen procesada, esto sólo se puede implementar con el filtro de umbralización, es representada mediante puntos aislados o bien representada por líneas horizontales. Tanto para el caso de puntos como para el de líneas, se representan los niveles de gris que hacen referencia al color negro.

Y por último, como opción complementaria a esta última, se añade un cursor mediante el cual el usuario puede determinar la separación entre los puntos o las líneas, en función de la opción escogida por el usuario. Cuanta menos separación se seleccione más real será la pintura representada, sin embargo se consumirán más recursos. Para un correcto uso de esta opción se añade un visor de vista previa de la imagen para que se haga la mejor elección.

7 Manual de usuario

En este capítulo se detallará un manual para el usuario, con el cual se podrá acceder a cualquier funcionalidad de la aplicación. Se explicarán detalladamente los paneles de cada una de los Forms y su función dentro del programa. También se hará una guía de botones con los que se ayudará al usuario a entender mejor el funcionamiento del sistema.

7.1 Estructura de los paneles

7.1.1 Robot Draw

Es la ventana principal del programa. Con ella se realiza la adquisición de la imagen por las distintas fuentes, se establece el filtro que se va a utilizar en cada caso y permite iniciar el proceso de envío de información.

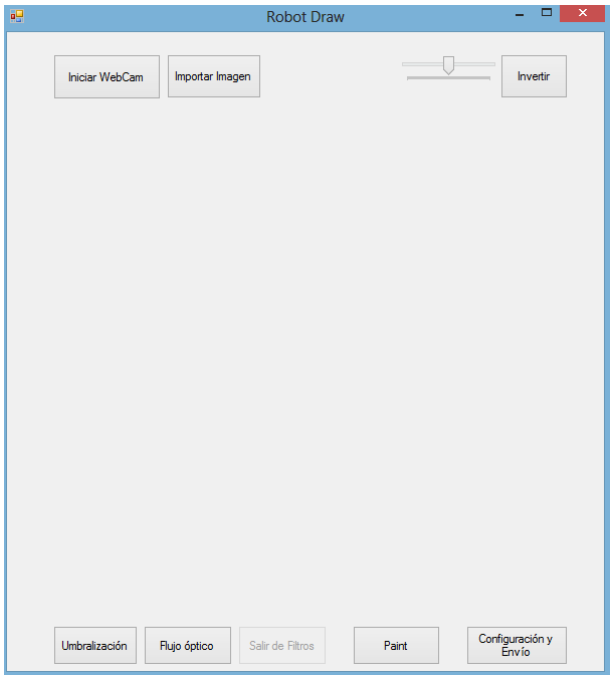


Figura 7.1 Ventana Robot Draw de la aplicación

7.1.2 VistaPrevia

Es la ventana dedicada a obtener una vista previa de la imagen para visualizar, antes de ser lanzada a la estación del robot. Esta ventana, si bien no es algo indispensable para el funcionamiento de la aplicación, es muy útil, ya que gracias a ella, se pueden ahorrar costes innecesarios.

Se permite configurar los parámetros de la imagen detallados en el capítulo de envío de información. Estos eran: tipo de muestreo, separación y tamaño del plano en el que se representa la imagen.

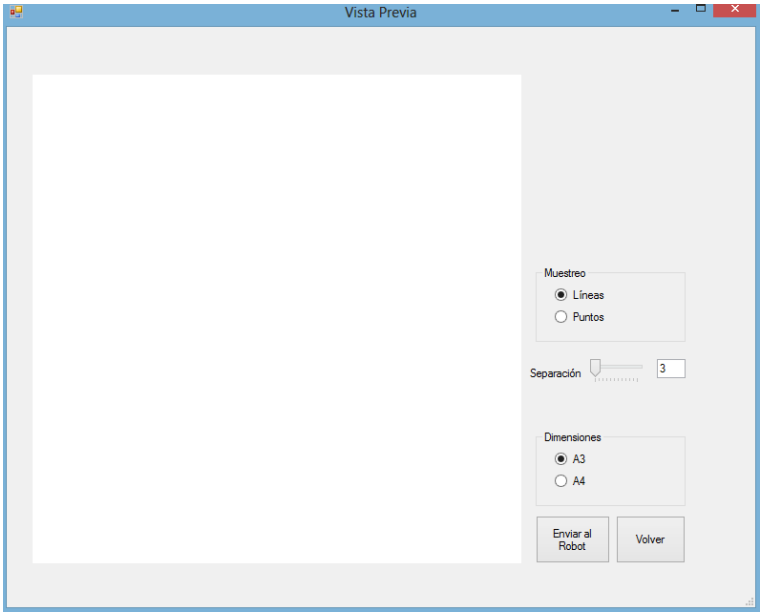


Figura 7.2 Ventana Vista Previa de la aplicación

Como se puede ver en la imagen, las opciones que se tiene por defecto son la representación en líneas, con una separación de 3 milímetros entre cada una de ellas y en el tamaño correspondiente al estándar DIN a3.

Esta ventana sólo será visible para el usuario una vez que ha adquirido la imagen que desea procesar o ha realizado el dibujo Paint. Si tras haber hecho esto la imagen resultante no es de agrado, se tiene la posibilidad de volver al panel anterior para repetir el procedimiento.

7.1.3 Position Control Panel

Esta ventana representa, en parte, lo que ve el usuario acerca del socket de comunicación. Contiene los controles para establecer la comunicación con la estación del robot y los mecanismos para mover, de forma individual, cada una de las articulaciones del robot, así como la libertad de mandarlo a cualquier posición XYZ admitida por los límites físicos del propio robot.

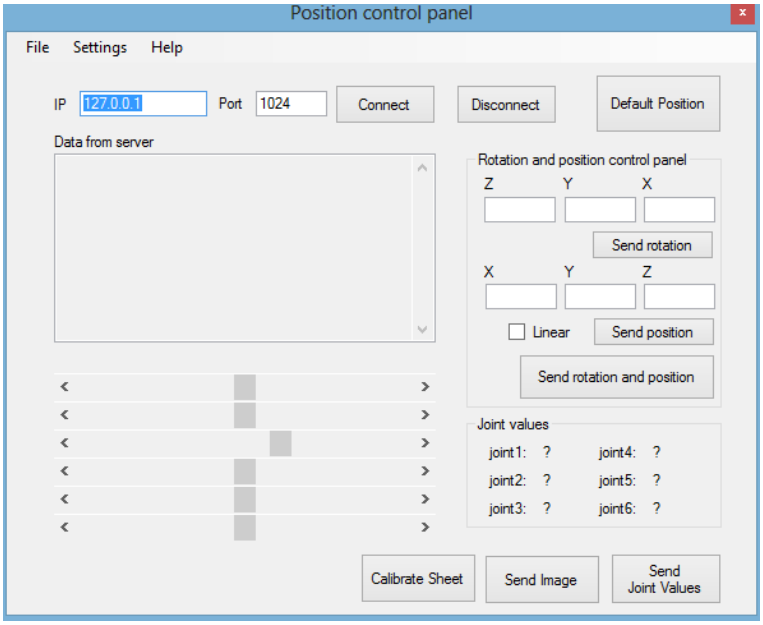


Figura 7.3 Ventana Position Control Panel de la aplicación

De esta forma quedan vistas las tres ventanas con las que puede interactuar el usuario, a continuación se puede ver mediante el siguiente esquema sencillo el orden de aparición.

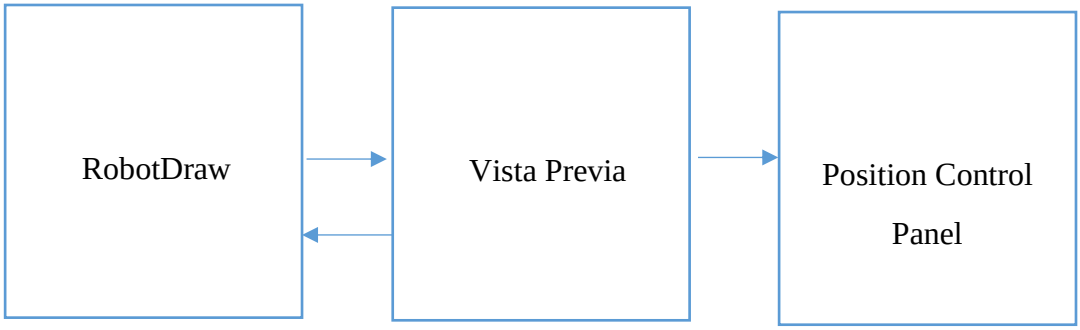


Figura 7.4 Orden aparición de ventanas interactivas

7.2 Guía de botones

Este punto servirá para conocer cada uno de los botones y la función que tienen en el propio programa en cada una de las ventanas disponibles detalladas en el capítulo anterior.

7.2.1 Botones RobotDraw

Iniciar Web Cam / Tomar Foto / Continuar adquiriendo 	Este es un único botón con tres estados dependiendo del punto en el que se esté ejecutando el programa: Iniciar WebCam: estado inicial del botón. Aparece al iniciar la aplicación, con este botón comienza la adquisición de imágenes desde la cámara web que esté conectada al equipo. Tomar Foto: estado del sistema en el que ya se ha iniciado la captura de imágenes y se ha seleccionado el filtro deseado para procesar la imagen de tal forma. Continuar Adquiriendo: estado siguiente a Tomar Foto, aparece cuando se ha tomado una foto. Ofrece la posibilidad de volver al estado anterior si la foto capturada no es satisfactoria
Importar Imagen 	Botón de un único estado. Presionando este botón se accede a un buscador interno en el que se pueden seleccionar archivos de imagen .jpg; .png; .bmp; .tiff. Se accede a una de las fuentes de adquisición planteadas en capítulos anteriores.

Cursor de Umbralización 	Con el selector de la figura se varía el valor del umbral del filtro umbralización, que por defecto es igual a 127. Se podrá oscilar entre los valores 0 y 255.
Invertir 	Mediante este botón, se invierte el funcionamiento del filtro umbral. El estado por defecto: Píxeles por debajo de 127 → a 0 y los superiores a 255. Este botón invierte ese estado.
Filtro de Umbralización 	Pulsando este botón, que sólo estará disponible cuando se esté adquiriendo imágenes por la cámara web o al importar una imagen, se le aplica a la imagen un filtro en el cual aparece representada con dos niveles de gris. 0 y 255.
Filtro Flujo Óptico 	Con este botón, la imagen adquirida aparece representada como vectores de posición entre aquellos píxeles móviles entre dos frames consecutivos.
Paint 	Acceso al panel donde se pueden hacer trazos libres, simulando un programa de dibujo
Salir de filtros 	Retorno a la adquisición de imágenes en formato RGB de la cámara web. Este botón solo está habilitado si se está utilizando un filtro.

Tabla 7.1 Botones disponibles en ventana Robot Draw

7.2.2 Botones Vista Previa

Muestreo 	Con la selección de una de estas opciones se determina si la representación de la imagen procesada por el filtro umbralización será mediante líneas horizontales o puntos.
Dimensiones 	Con la selección de una de estas opciones se determina el tamaño en el cual se va a representar la imagen. Es válido para todos los tipos de adquisición.
Separación 	Con este cursor se puede variar el espacio de separación entre líneas o puntos de la representación de la imagen. Unidades : milímetros. Indicador (valor 3 por defecto)
Enviar al Robot 	Botón de envío de la información. Haciendo clic en este botón se accede a la ventana de conexión, donde se procede al envío definitivo de la información.

Tabla 7.2 Botones disponibles en ventana Vista Previa

7.2.3 Botones Position Control Panel

Esta ventana interactiva es parte del programa proporcionado "Socket based communication in RobotStudio for controlling ABB-IRB120 robot. Design and development of a palletizing station", con lo que no se va a detallar el funcionamiento de los botones que ya estaban implementados, sólo se explicarán los botones posteriormente.

Calibrate Sheet 	Botón calibrado de folio. Al hacer clic en este botón se manda una información a la estación del robot correspondiente a las dimensiones elegidas en la ventana de configuración para representar el dibujo.
Send Image 	Botón de envío definitivo de la información. Haciendo clic en este botón la información de la imagen o Paint se ordena al brazo robotizado para que la represente.

Connect 	Este botón corresponde al socket proporcionado pero es fundamental señalar que, tras haber ejecutado el programa en la estación del robot, se debe pulsar este botón para que los botones del panel conexión tengan efecto en el sistema.

Tabla 7.3 Botones disponibles en ventana Position Control Panel

7.3 Flujo de aplicación

Ya detallados los paneles y sus botones, se puede realizar un pequeño manual en el que se demuestra cada uno de los flujos de trabajo.

7.3.1 Flujo en la umbralización

7.3.1.1 Adquisicíon desde Web Cam

Se inicia el flujo con la ventana RobotDraw

1. Clic ”Iniciar WebCam”

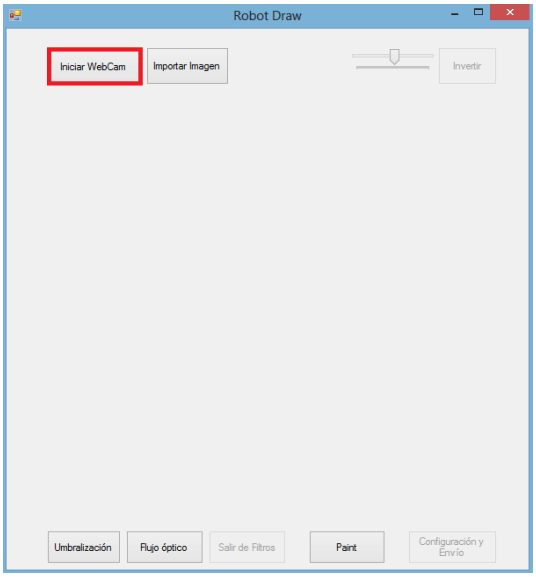


Figura 7.5 Paso Iniciar Web Cam en Flujo de umbralización

2. Clic ”Umbralización”
3. Con botón ”Invertir” y ”cursor” → Variación del efecto
4. Clic ”Tomar Foto”
5. Clic ”Configuración y envío”

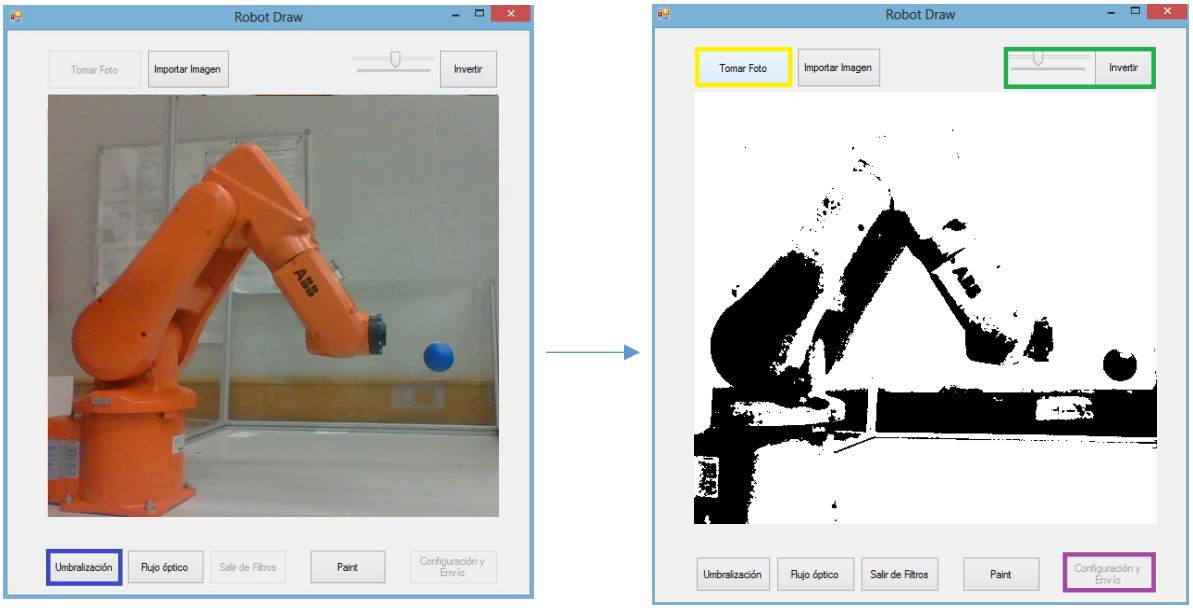


Figura 7.6 Umbralización y captura en Flujo de umbralización

7.3.1.2 Adquisición desde el propio equipo

Se inicia el flujo con la ventana RobotDraw

1. Clic ”Iniciar WebCam”

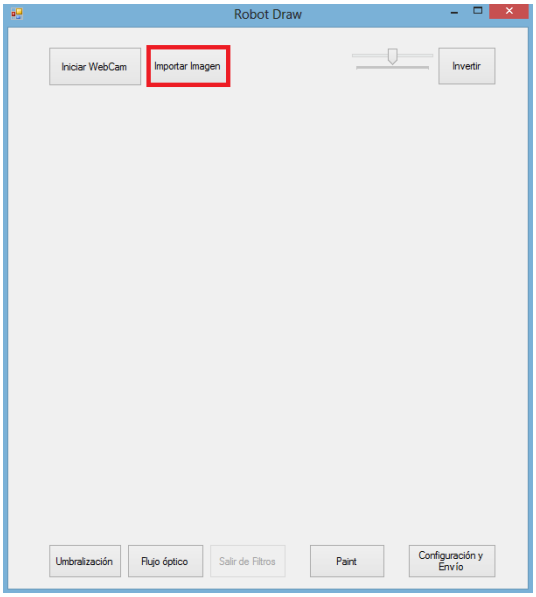


Figura 7.7 Paso Importar Imagen en Flujo de umbralización

2. Clic ”Umbralización”
3. Con botón ”Invertir” y ”cursor” → Variación del efecto
4. Clic ”Configuración y envío”

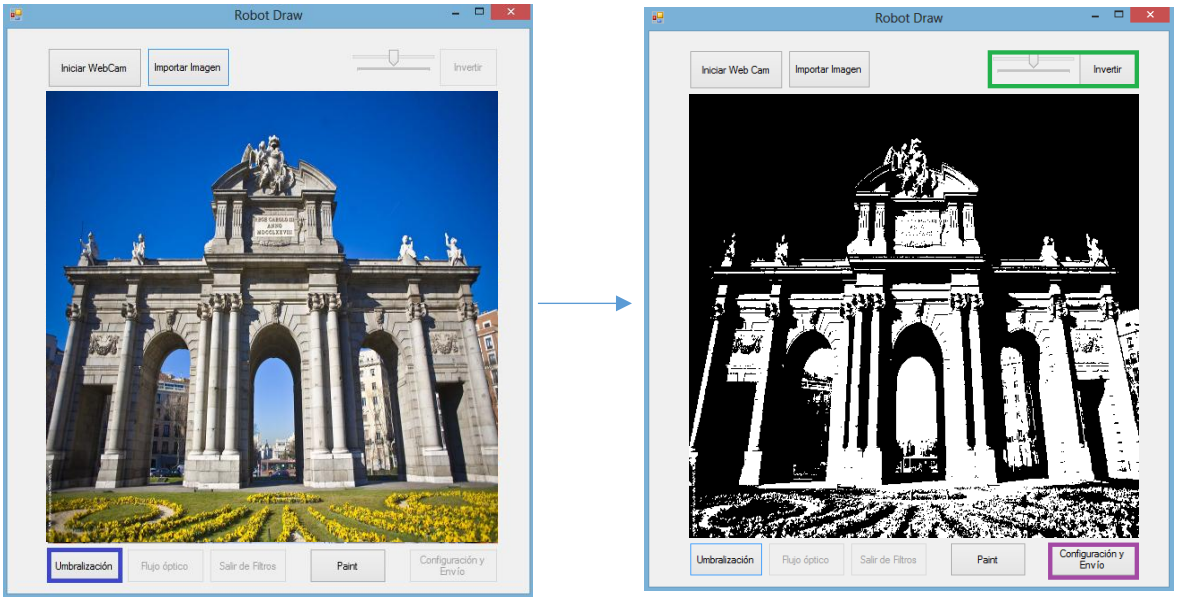


Figura 7.8 Imagen importada y umbralizada en Flujo de umbralización

7.3.1.3 Configuración y Envío

Después de seguir los pasos detallados en los capítulos de adquisición, ahora se procede a seleccionar los parámetros de configuración de la imagen, en la ventana vista previa, que son: el tipo de representación, su separación y las dimensiones del plano en el que se representará la imagen. Para ello:

1. En "Muestreo", seleccionar la opción deseada
2. Con el cursor, determinar la separación elegida para realizar la representación
3. En "Dimensiones", seleccionar la opción deseada
4. Clic en "Enviar al Robot" si la vista previa es satisfactoria

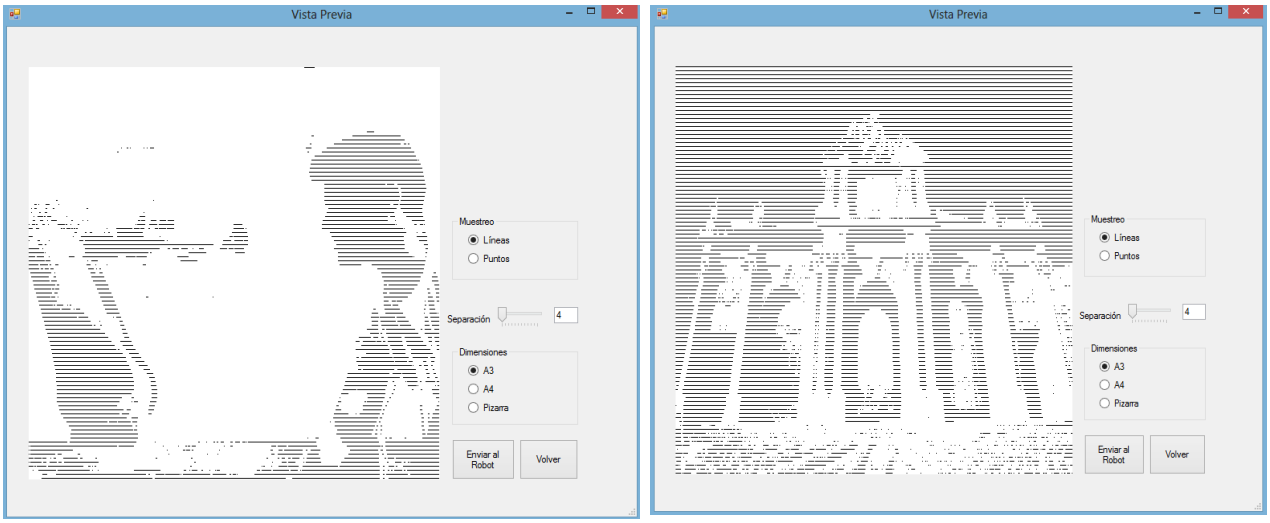


Figura 7.9 Configuración de las imágenes procesadas en Flujo de umbralización

Tras este último paso, se abrirá la ventana de conexión con la que se realiza la conexión con la estación de trabajo del robot.

1. Clic en "Connect". (Previamente iniciar simulación Robot)
2. Clic en "Calibrate Sheet" para que el robot marque los límites del folio
3. Clic en "Send Image" para comenzar la representación

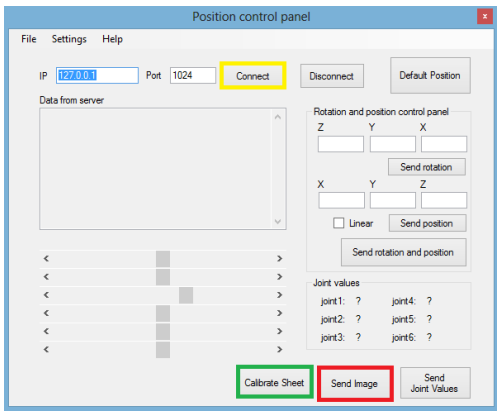


Figura 7.10 Envío de las imágenes procesadas en Flujo de umbralización

7.3.2 Flujo en flujo óptico

Se inicia el flujo con la ventana RobotDraw

1. Clic ”Iniciar Web Cam”

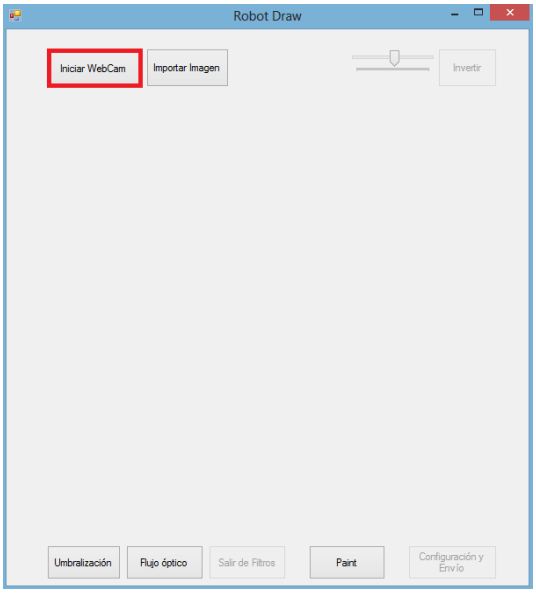


Figura 7.11 Paso Iniciar Web Cam en Flujo óptico

2. Clic ”Flujo óptico”
3. Clic ”Tomar Foto”
4. Clic ”Configuración y envío”

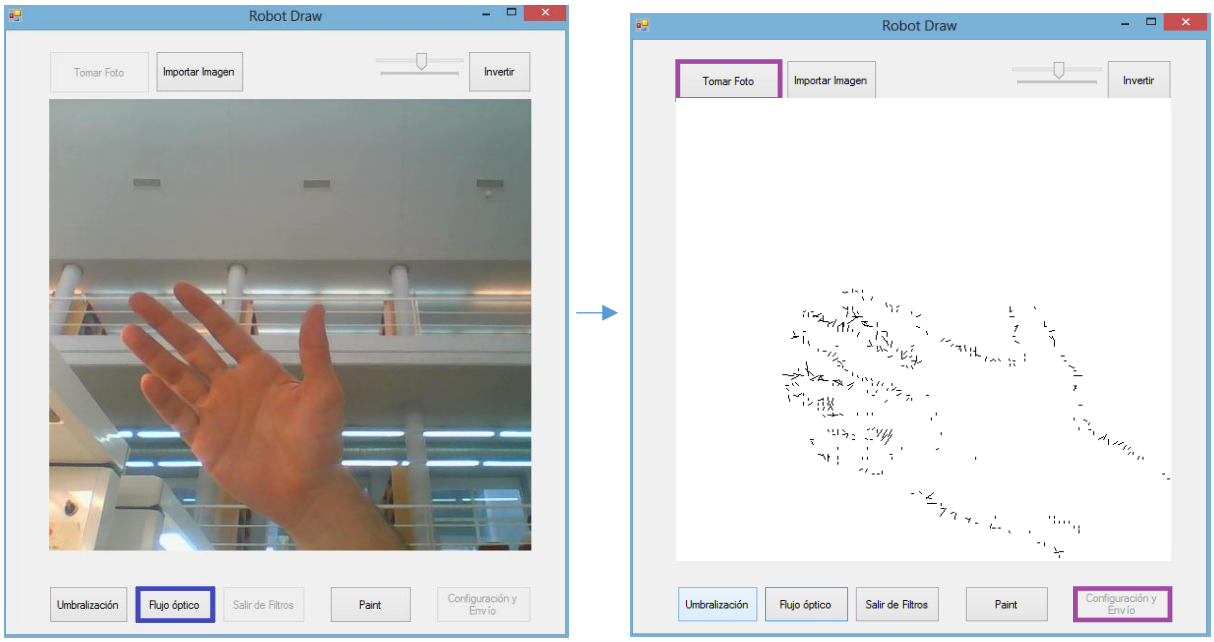


Figura 7.12 Imagen adquirida y convertida a vectores de posición en Flujo óptico

7.3.2.1 Configuración y Envío

Después de seguir los pasos detallados en los capítulos de adquisición, ahora se procede a seleccionar los parámetros de configuración de la imagen, en la ventana vista previa, que son: el tipo de representación, su separación y las dimensiones del plano en el que se representará la imagen. Para ello:

1. En "Dimensiones", seleccionar la opción deseada
2. Clic en "Enviar al Robot" si la vista previa es satisfactoria

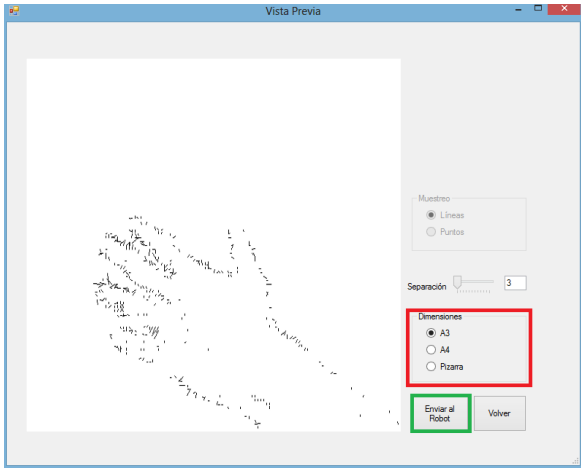


Figura 7.13 Configuración de la imagen procesada en Flujo óptico

Tras este último paso, se abrirá la ventana de conexión con la que se realiza la conexión con la estación de trabajo del robot.

1. Clic en "Connect", después de haber iniciado la ejecución del software del robot
2. Clic en "Calibrate Sheet" para que el robot marque los límites del folio
3. Clic en "Send Image" para comenzar la representación

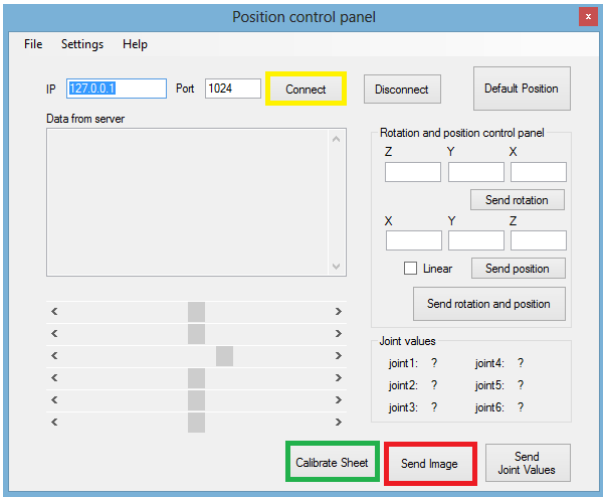


Figura 7.14 Envío de imagen procesada en Flujo óptico

7.3.3 Flujo en aplicación Paint

7.3.3.1 Dibujo Paint

Se inicia el flujo con la ventana RobotDraw

1. Clic ”Paint”

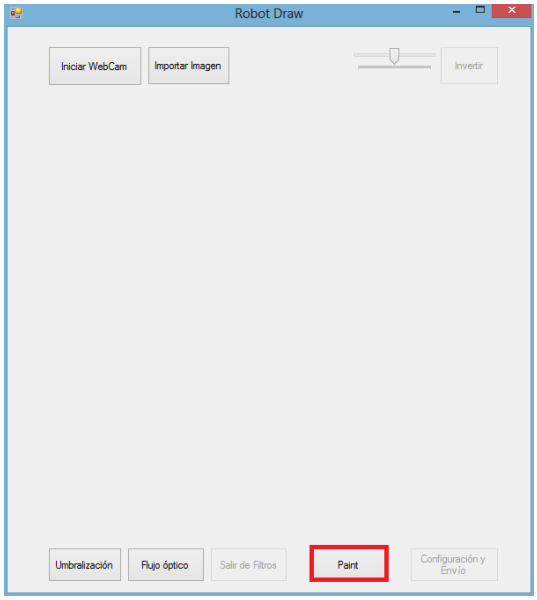


Figura 7.15 Paso Iniciar proceso Paint

2. Libre trazado por el panel
3. Clic ”Configuración y envío”

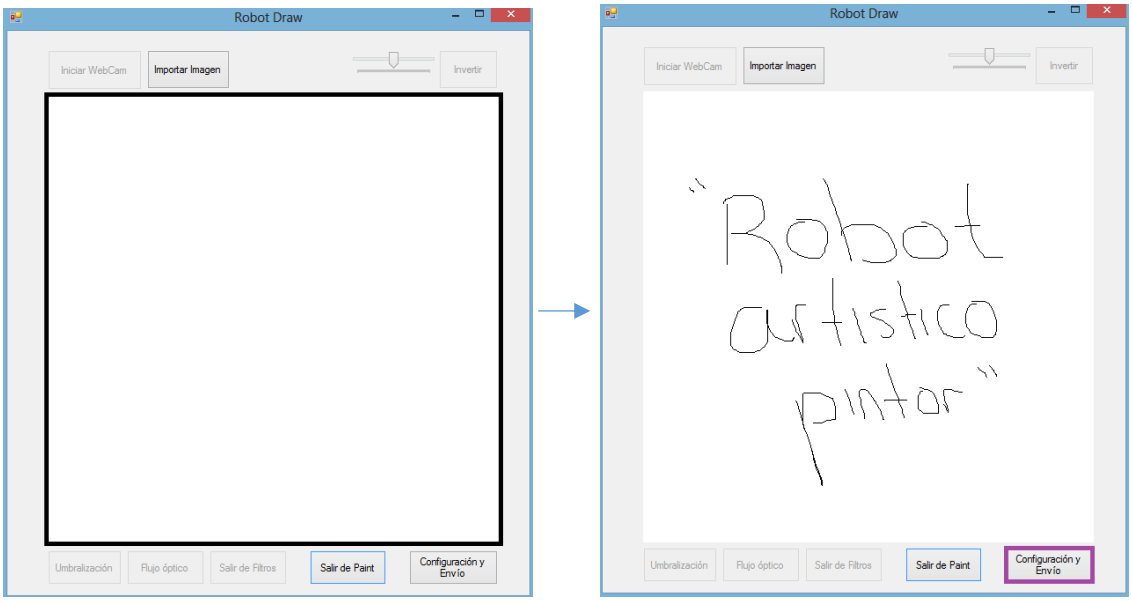


Figura 7.16 Trazado de dibujo en proceso Paint

7.3.3.2 Configuración y Envío

Después de seguir los pasos detallados en el capítulo de dibujo, ahora se procede a seleccionar el parámetro de configuración de la imagen, en la ventana vista previa, que son las dimensiones del plano en el que se representará la imagen. Para ello:

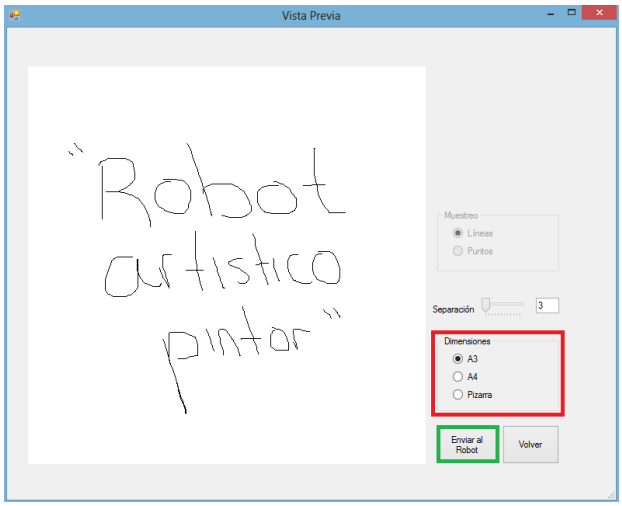


Figura 7.17 Configuración del dibujo en proceso Paint

Tras este último paso, se abrirá la ventana de conexión con la que se realiza la conexión con la estación de trabajo del robot.

1. Clic en "Connect", después de haber iniciado la ejecución del software del robot.
2. Clic en "Calibrate Sheet" para que el robot marque los límites del folio
3. Clic en "Send Image" para comenzar la representación

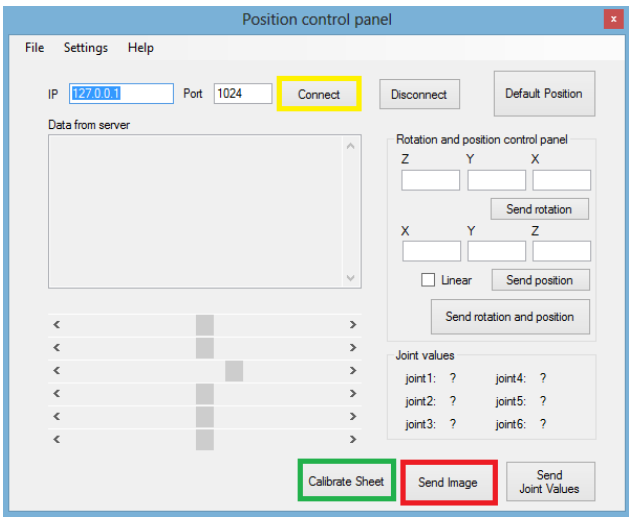


Figura 7.18 Envío del dibujo en proceso Paint

8 Resultados

8.1 Introducción

En este capítulo se presentarán los resultados obtenidos tras haber realizado de manera práctica todos los casos citados en capítulos previos. Para ello, y como ya se estableció, se realizan dos comunicaciones. La primera opción es la realización de los ejemplos en el software RobotStudio, de manera virtual. Y la segunda opción constituye la realización de los ejemplos con la estación de trabajo real del robot, compuesta por el brazo robótico ABB IRB -120 y el controlador IRC5.

Tanto para el software como para el propio brazo robótico, se utiliza el mismo modelo ya que el software recrea el robot y su estación. Por lo tanto, es bueno conocer mejor el robot y la herramienta con la que trabaja.

8.2 Espacio de trabajo

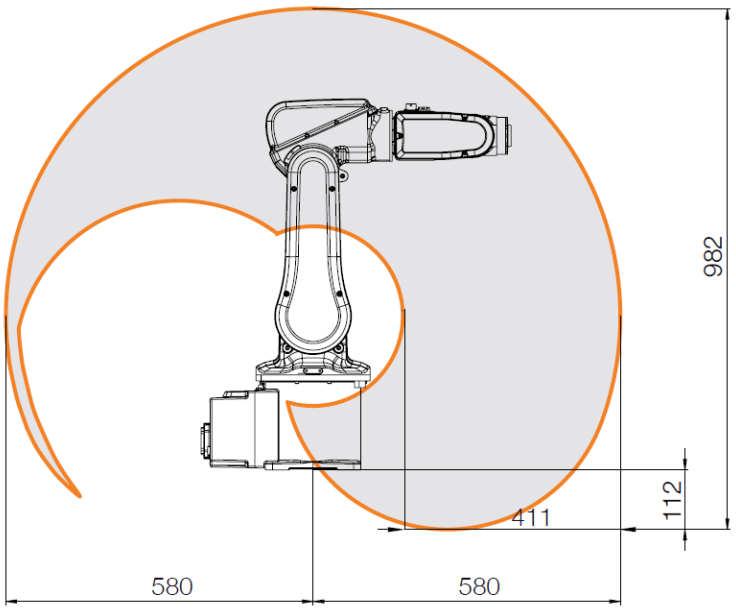


Figura 8.1 Espacio de trabajo – vista lateral

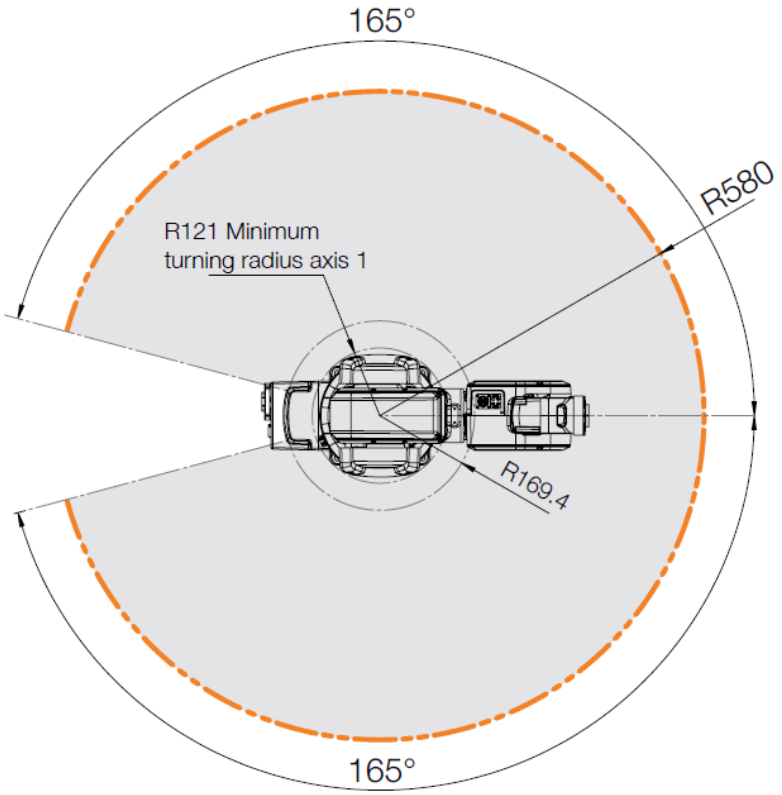


Figura 8.2 Espacio de trabajo – vista superior

Este espacio de trabajo en el que se puede mover el robot es fundamental tenerlo en cuenta a la hora de controlar los límites de las diferentes tipos de dimensiones que se permiten para la representación.

8.3 Herramienta Utilizada

Otro punto fundamental, es establecer la herramienta virtual con la que se va a simular el trazado en el software o la herramienta real con la que el robot va a representar las imágenes que le son enviadas.

Esta herramienta está basada en una práctica realizada en la asignatura "Sistemas Robotizados" en la que se diseñó una herramienta con un propósito similar. Se le ha asignado el nombre de Porta_rotulador, ya que simula una estructura en la cual se puede fijar un rotulador que pinte cuando está en contacto con el plano.

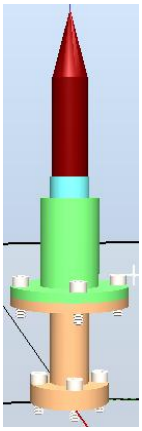


Figura 8.3 Herramienta Porta_Rotulador

8.4 Representación en RobotStudio

Para la representación de las imágenes en RobotStudio se ha diseñado una estación de trabajo virtual. Esta estación está basada en el trabajo "Socket based communication in RobotStudio for controlling ABB-IRB120 robot. Design and development of a palletizing station" de Marek Frydrysiak, por lo que no se entrará en detalle. La interface de la estación es la que a continuación se presenta. En ella se puede distinguir el brazo robótico, una estructura de seguridad y la herramienta Porta_Rotulador citada en el punto anterior.

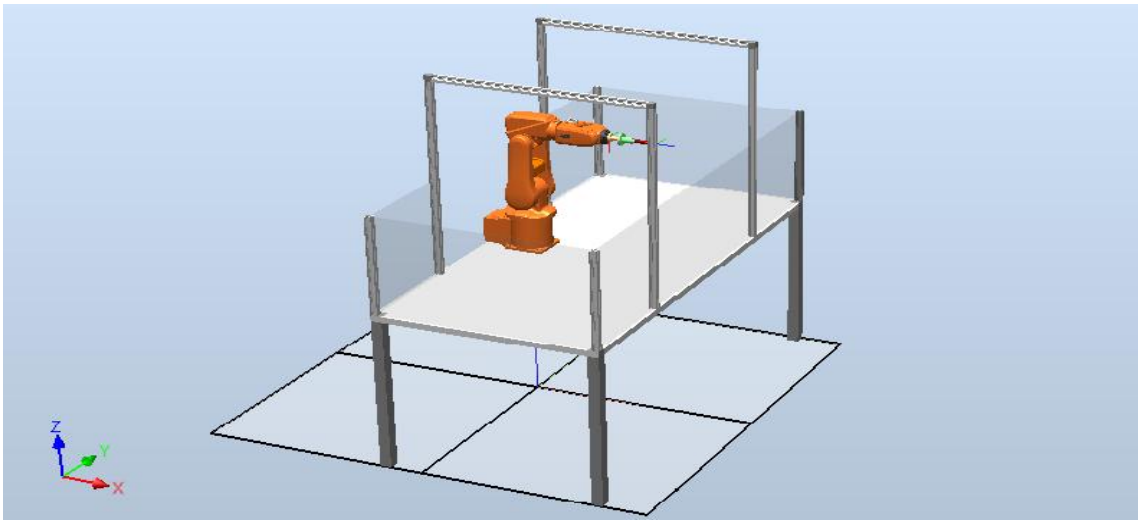


Figura 8.4 Interface principal de la estación de trabajo en RobotStudio

8.4.1 Manual de usuario

Antes de mostrar los casos prácticos correspondientes a este apartado, se va a realizar un pequeño manual con el que el usuario sea capaz de representar estos mismos ejemplos de manera autónoma.

Una vez abierta la estación virtual con RobotStudio, aparecerá la interface principal que se ha mostrado y su barra de acciones.

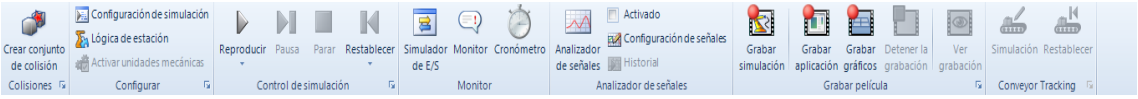


Figura 8.5 Menú - Robotstudio

Para que los motores del robot pasen a un estado de encendido y por tanto el robot esté habilitado para su funcionamiento, se debe hacer clic en "Reproducir" (Este paso debe ser realizado antes de solicitar la conexión desde la aplicación de procesamiento de imágenes).

Además, para una correcta visualización de la trayectoria seguida por el robot, se puede habilitar una opción para tal propósito. Para ello, se hace clic en "Monitor" y se establece el color de la traza y la longitud de la misma. Se recomienda establecer una longitud elevada para que se contemple toda la imagen.

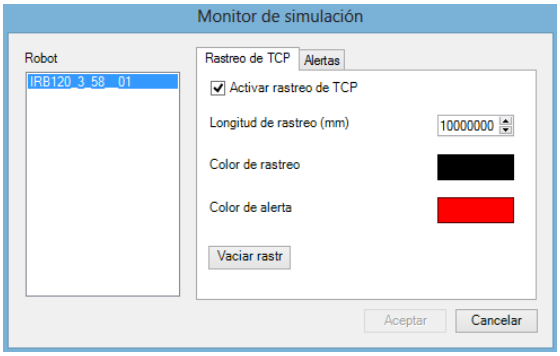
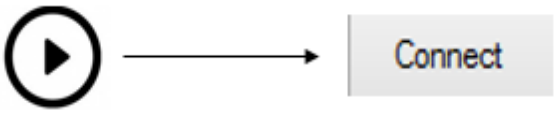


Figura 8.6 Monitor de simulación - RobotStudio

Una vez seguidos estos pasos, si ya se ha realizado el procesamiento de la imagen con la aplicación que se ha desarrollado, el robot está preparado para iniciar la recepción de información. Como se ha mencionado, antes de hacer clic en el botón "connect" de la ventana "Position control panel" se ha de iniciar la simulación en la estación virtual:



8.4.2 Resultados prácticos obtenidos

8.4.2.1 Resultados imagen importada

En este apartado se ofrece una colección de imágenes referentes a cada uno de los casos teóricos que ofrece la aplicación.

Se comienza con una imagen cualquiera importada desde el equipo, la cual se va a representar mediante puntos y líneas.

Figura 8.7 Resultados -Imagen importada desde el equipo

Esta es la imagen que se ha utilizado como ejemplo para realizar la representación del caso "Imagen importada". Esta imagen será procesada mediante los pasos detallados en capítulos anteriores hasta llegar a los resultados que a continuación se presentan.

Se presentan dos casos para cada tipo de representación de líneas y puntos. El primer caso con una separación en el muestreo de 4mm y el segundo caso con una separación de 10mm, de esta forma se puede apreciar de forma significativa el cambio. Además se incluye capturas de la ventana configuración en cada caso, con lo que quedan detallados los parámetros de configuración.

8.4.2.1.1 Representación imagen importada mediante líneas

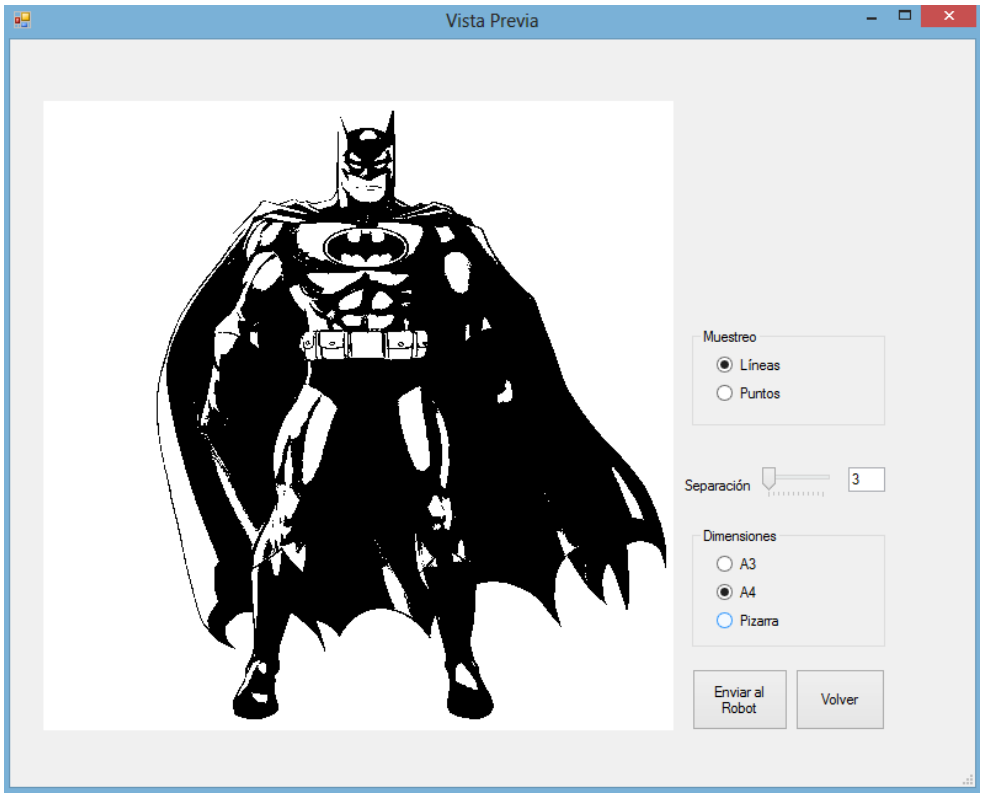


Figura 8.8 Resultados - Imagen importada vista previa - umbralizada

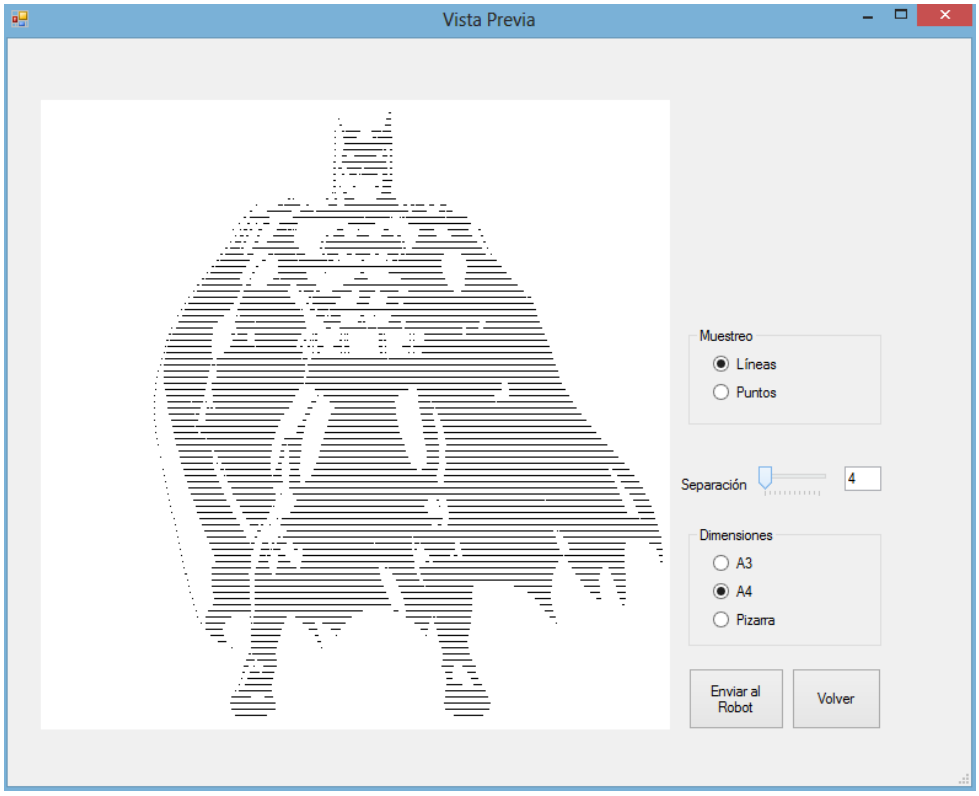


Figura 8.9 Resultados – Imagen importada vista previa – líneas 4 mm

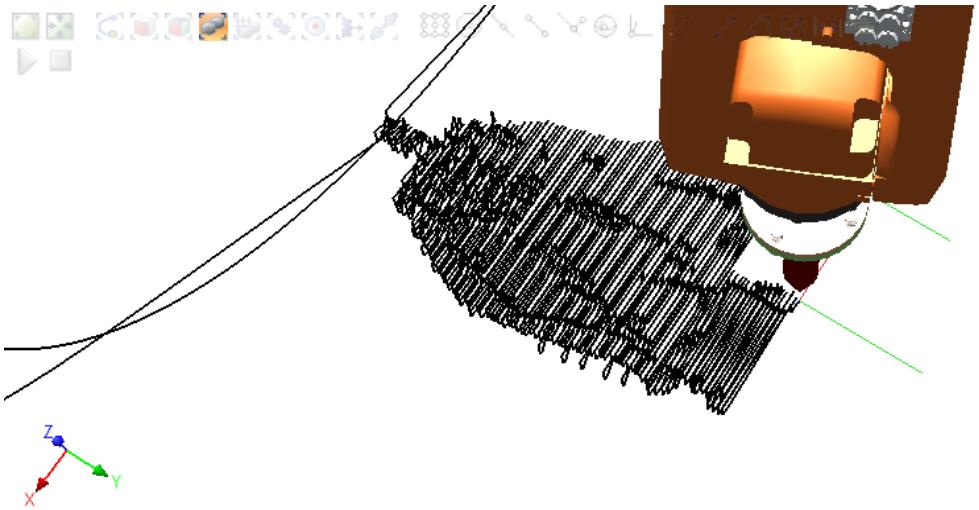


Figura 8.10 Resultados - Representación imagen importada vista superior – Líneas 4mm

Figura 8.11 Resultados - Representación imagen importada vista angulada – Líneas 4mm

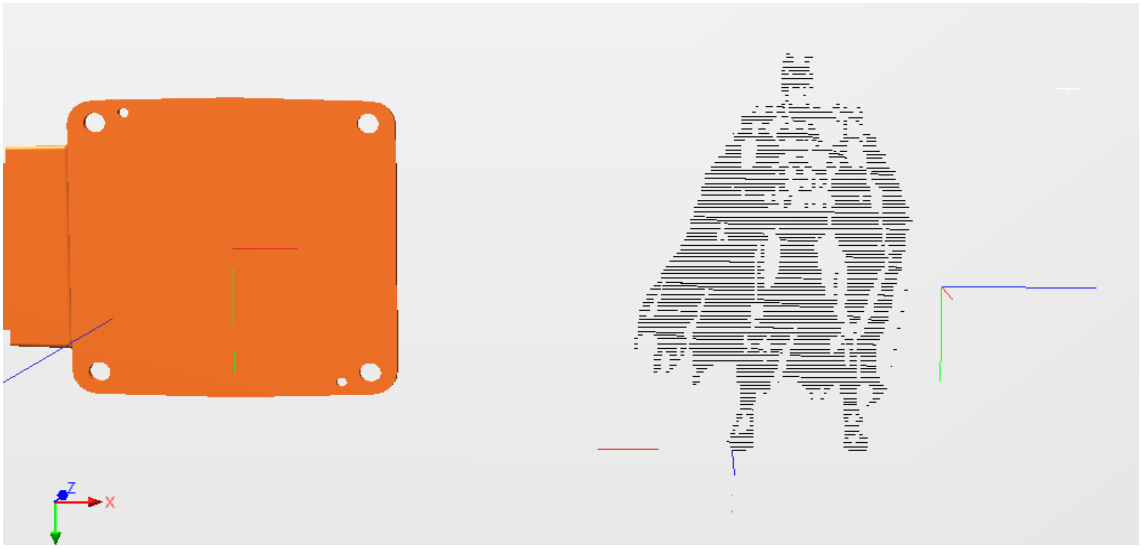


Figura 8.12 Resultados – Representación imagen importada vista inferior – Líneas 4mm

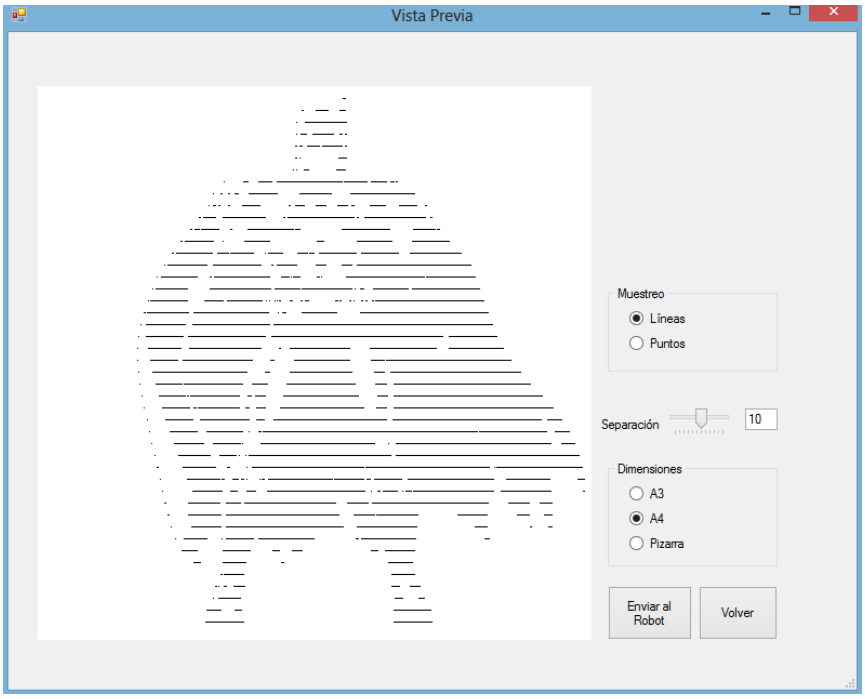


Figura 8.13 Resultados - Imagen importada vista previa – líneas 10 mm

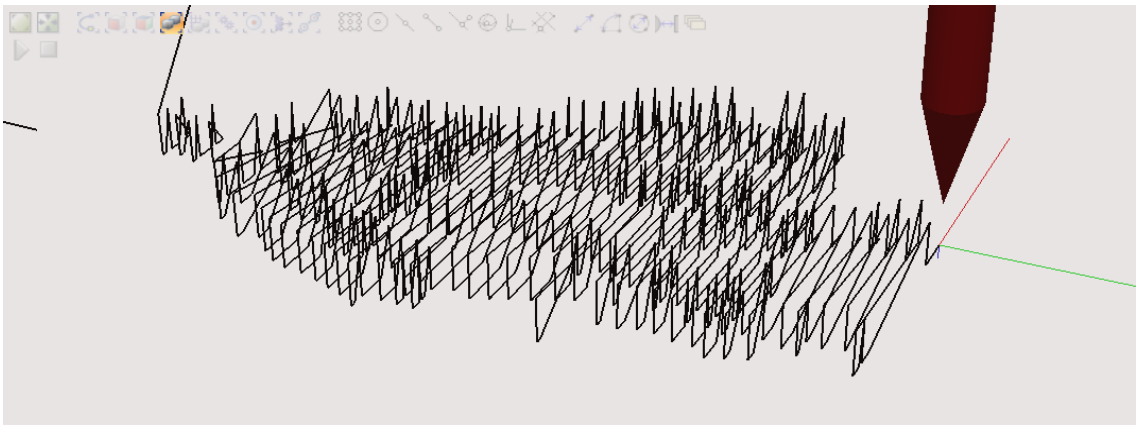


Figura 8.14 Representación – Imagen importada vista superior – Líneas 10 mm

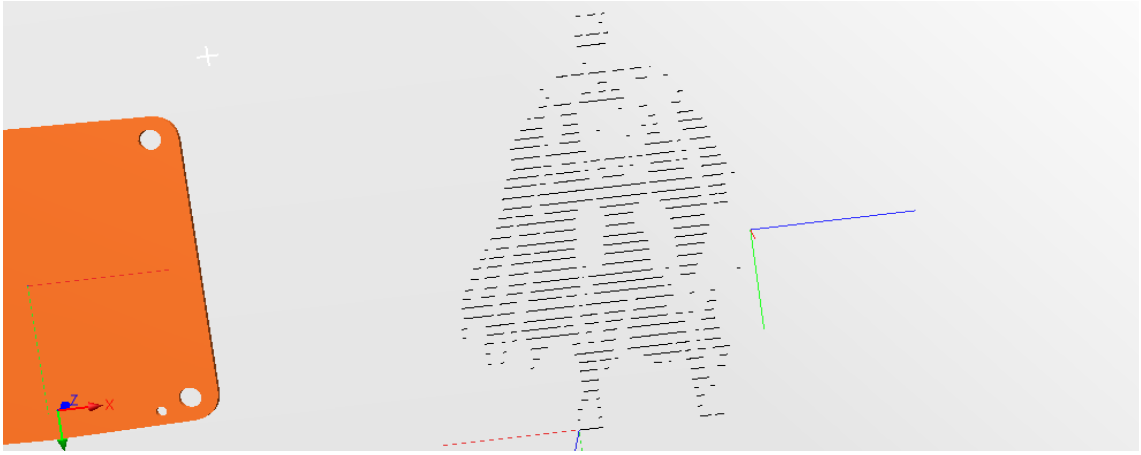


Figura 8.15 Resultados – Imagen importada vista inferior – Líneas 10 mm

8.4.2.1.2 Representación imagen importada mediante puntos

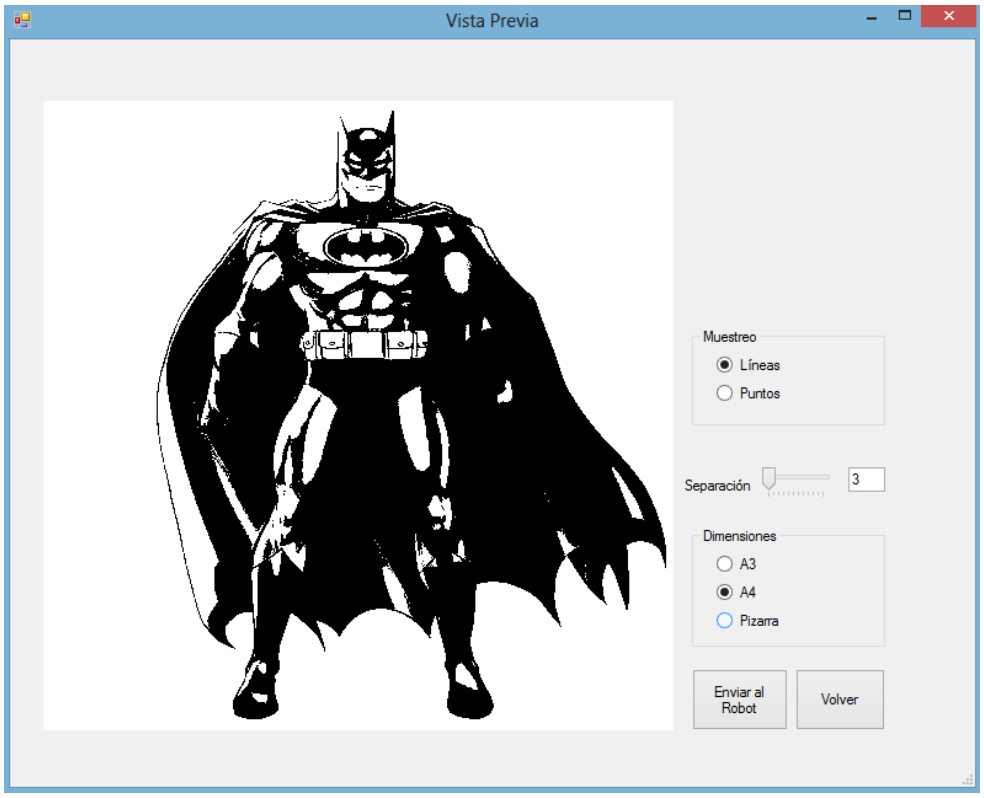


Figura 8.16 Resultados - Imagen importada vista previa - umbralizada

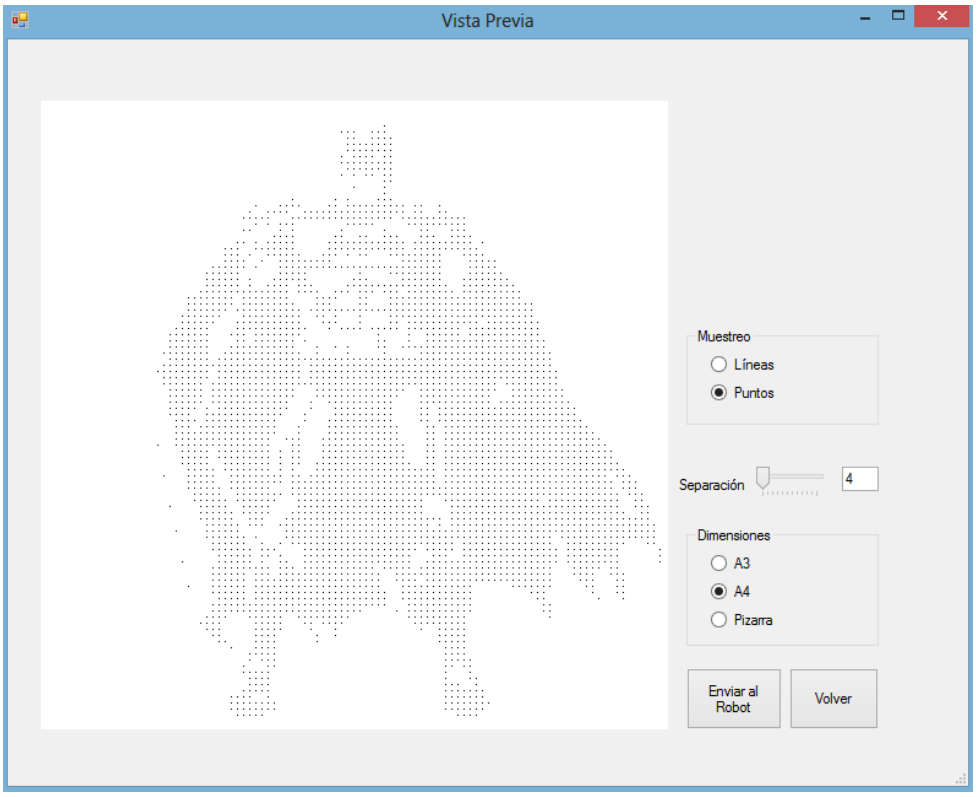


Figura 8.17 Resultados – Imagen importada vista previa – puntos 4mm

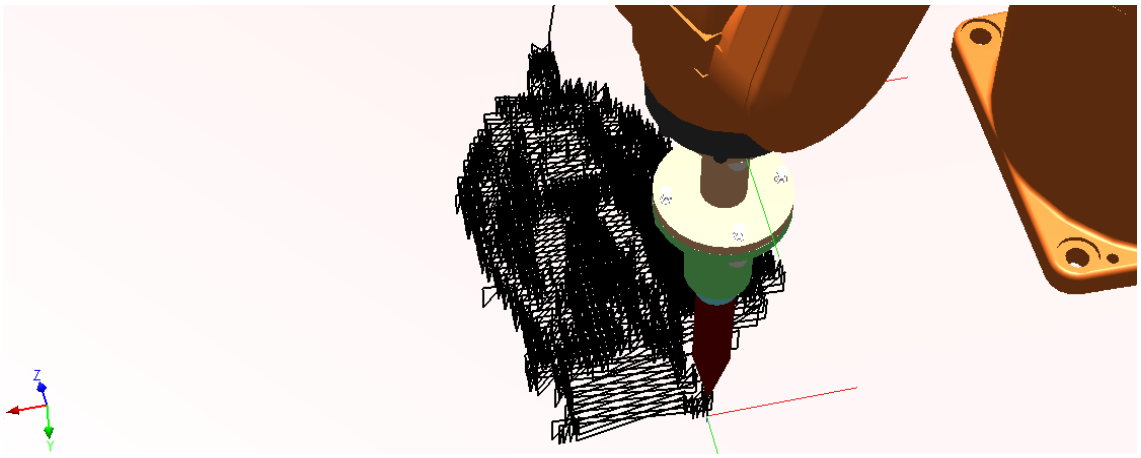


Figura 8.18 Resultados - Representación imagen importada vista superior – Puntos 4 mm

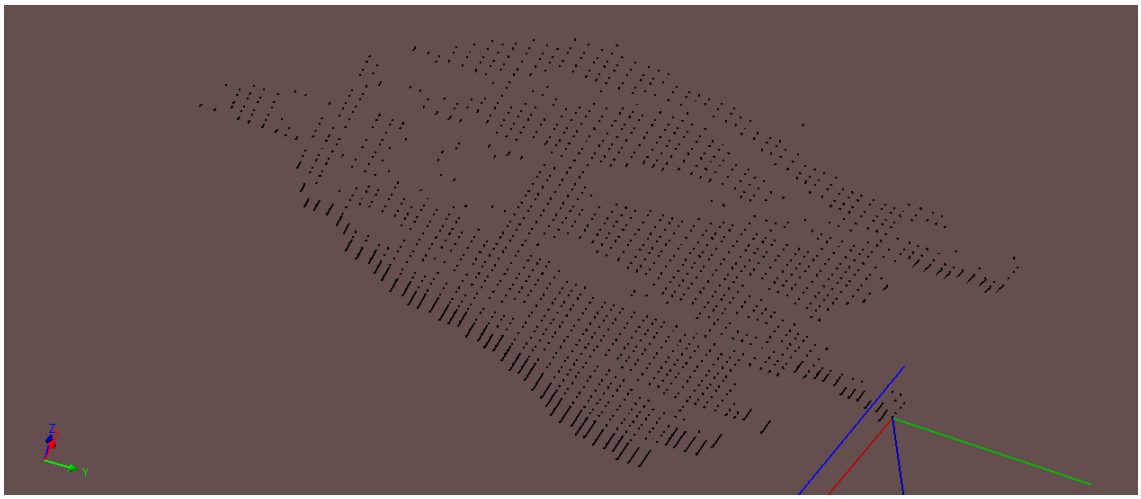


Figura 8.19 Resultados - Representación imagen importada vista angulada – Puntos 4 mm

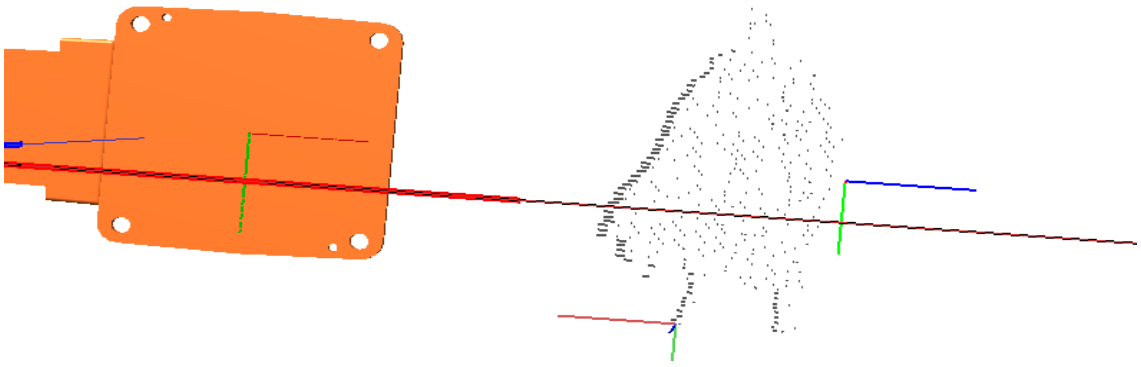


Figura 8.20 Resultados - Representación imagen importada vista inferior – Puntos 4 mm

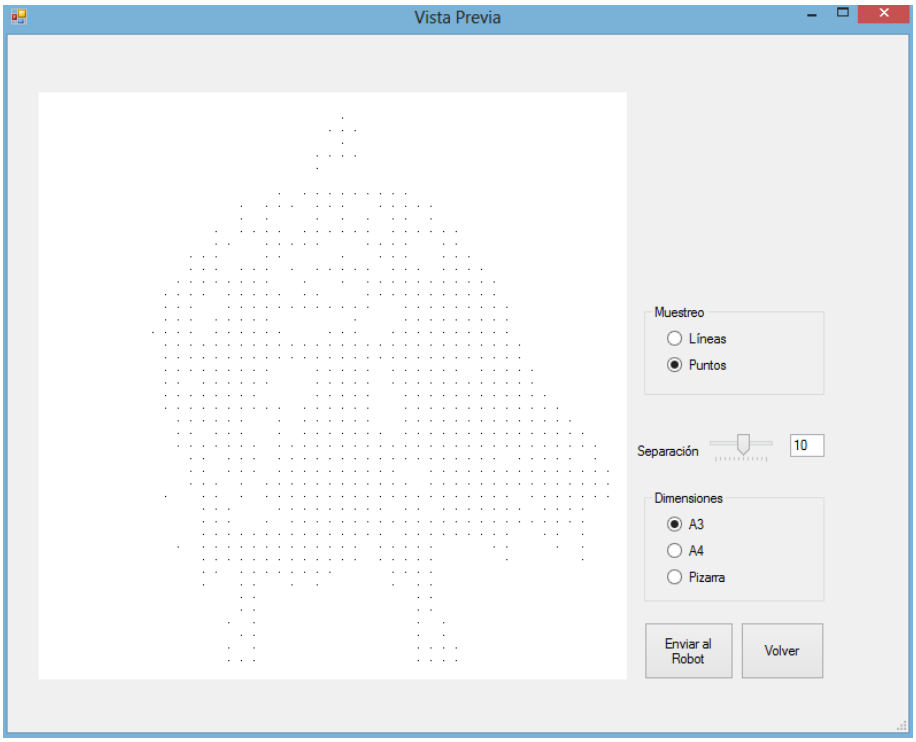


Figura 8.21 Resultados - Imagen importada vista previa – Puntos 10 mm

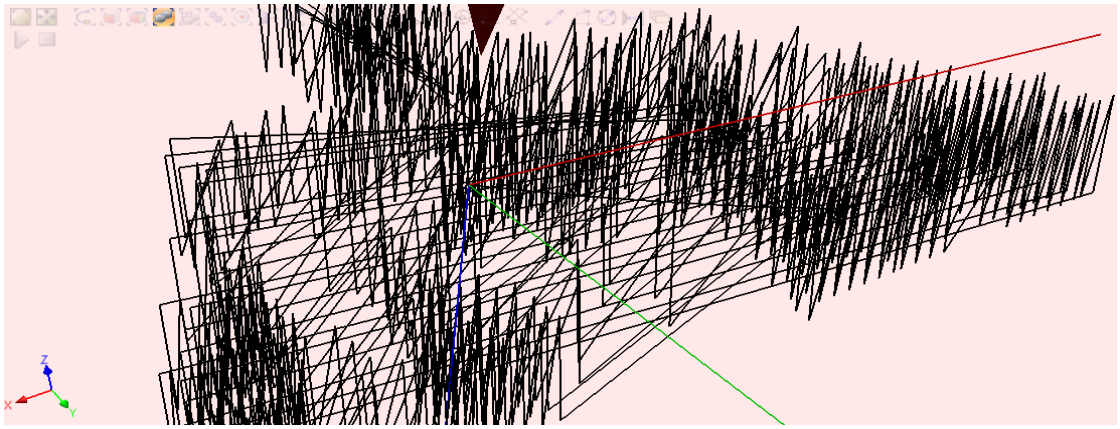


Figura 8.22 Resultados - Imagen importada vista superior – Puntos 10 mm

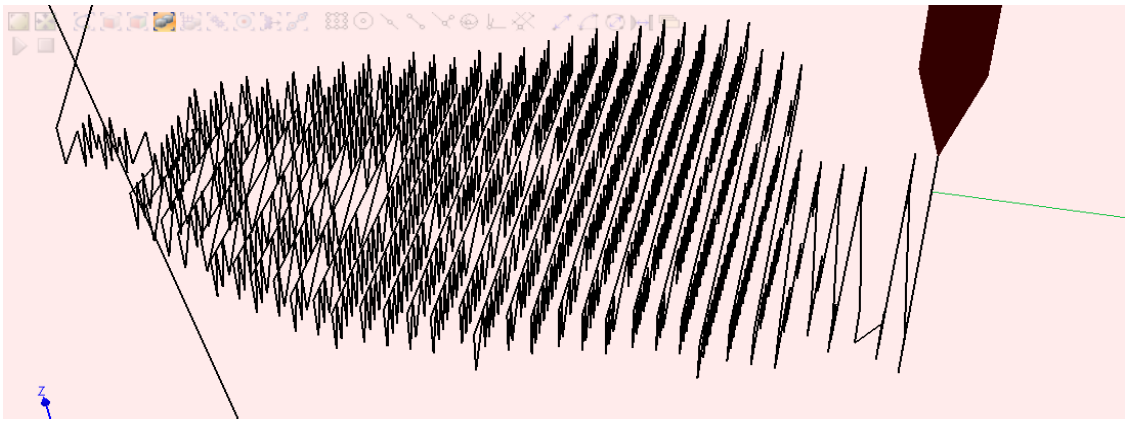


Figura 8.23 Resultados – Imagen importada vista angulada – Puntos 10 mm

8.4.2.2 Resultados aplicación Paint

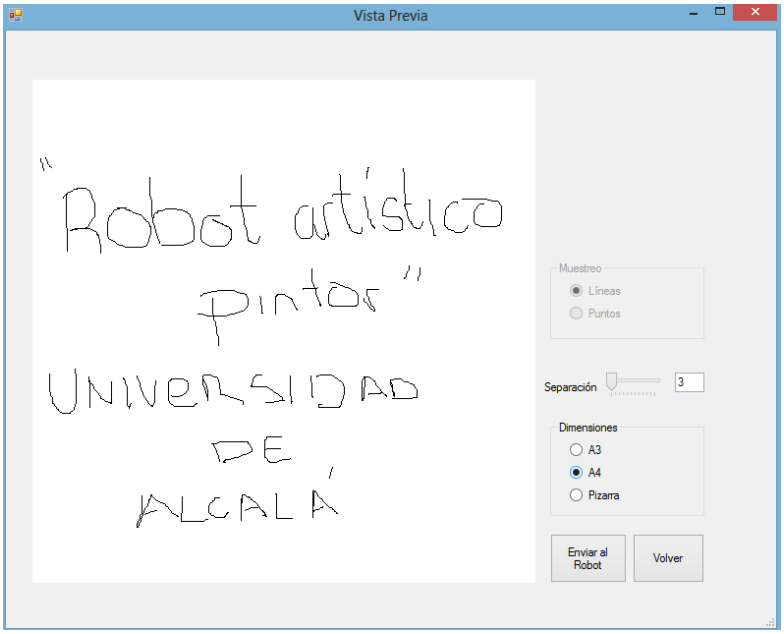


Figura 8.24 Resultados - Imagen Paint vista previa

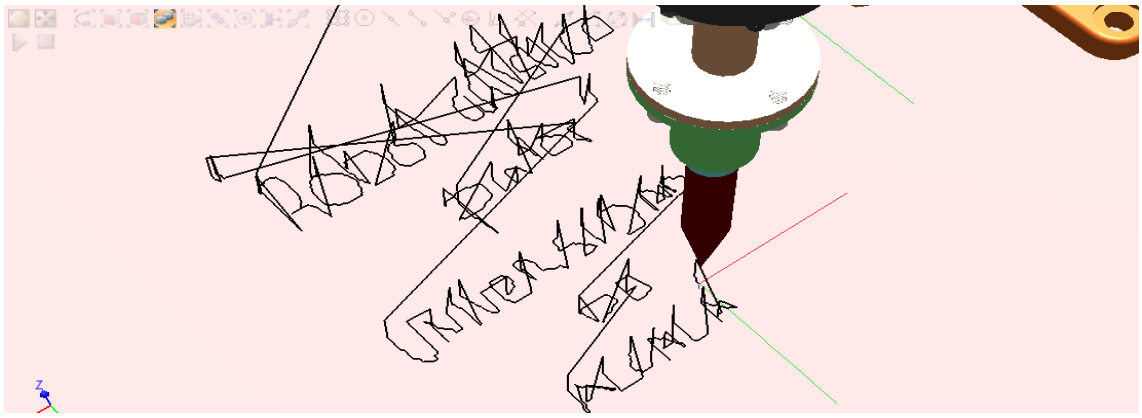


Figura 8.25 Resultados - Imagen Paint vista superior

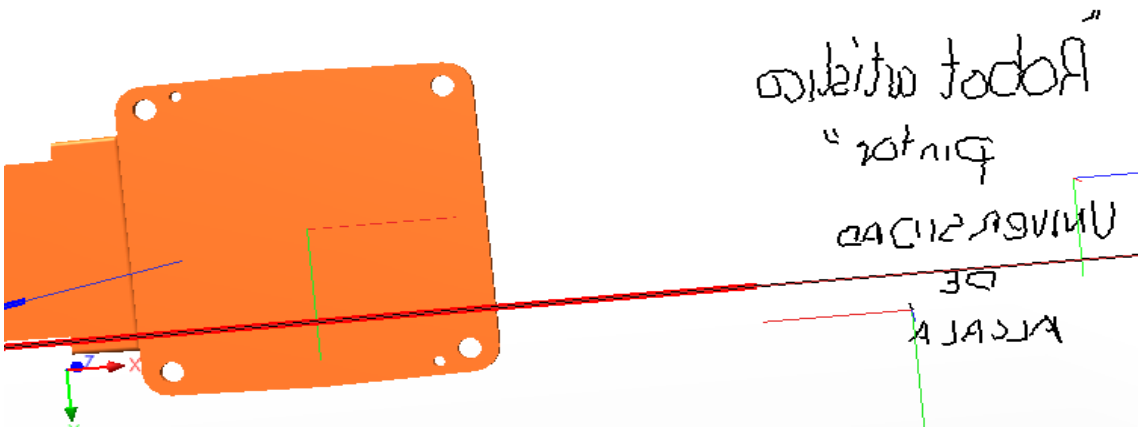


Figura 8.26 Resultados - Imagen Paint vista inferior

8.4.2.3 Resultados adquisición cámara web – flujo óptico

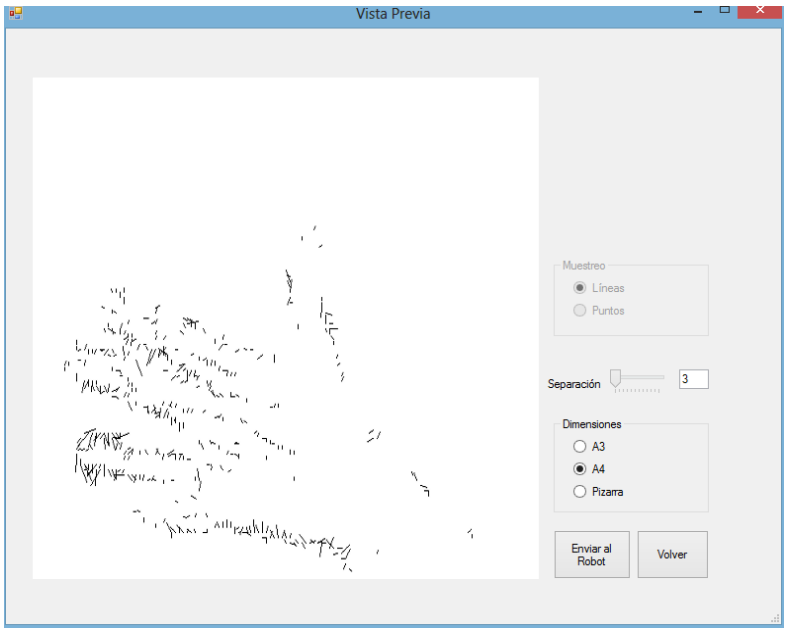


Figura 8.27 Resultados - Imagen Flujo óptico en vista previa

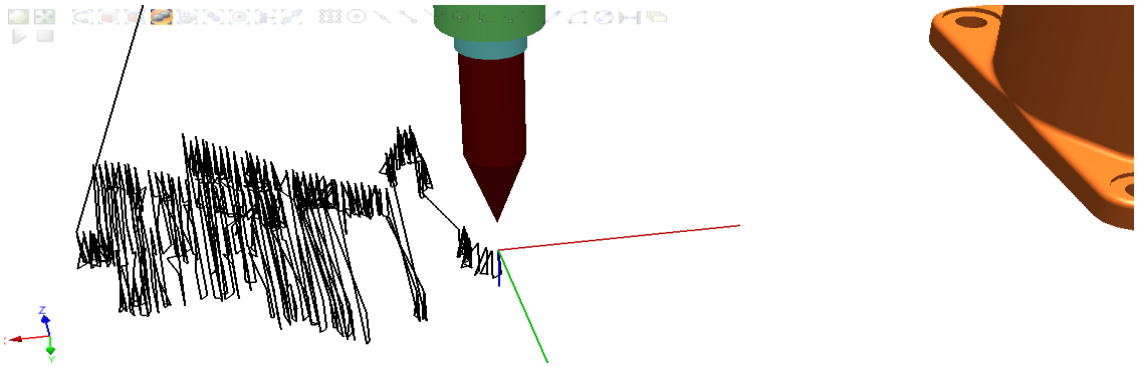


Figura 8.28 Resultados - Imagen Flujo óptico vista superior

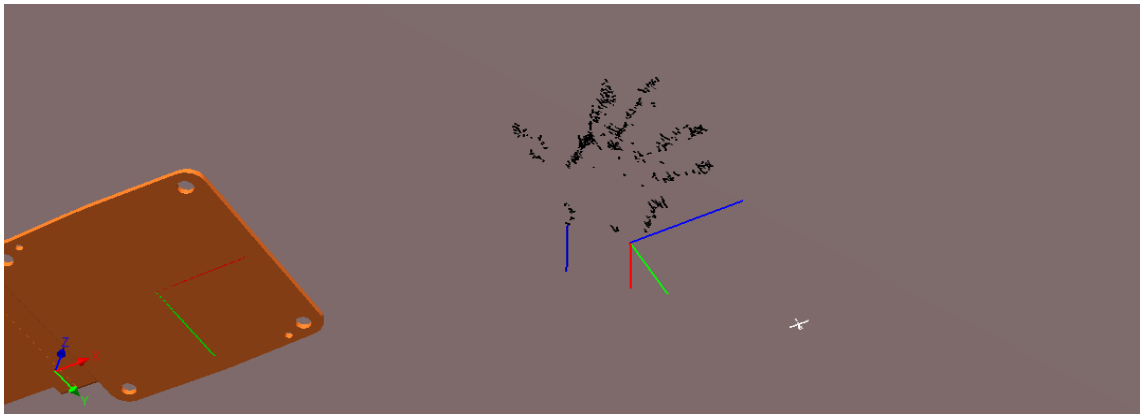


Figura 8.29 Resultados - Imagen Flujo óptico vista inferior

8.4.3 Resultados adquisición cámara web - umbralización

Para finalizar la demostración de los casos a representar en RobotStudio se adquiere una imagen desde la cámara web del equipo, para representarla mediante puntos y líneas

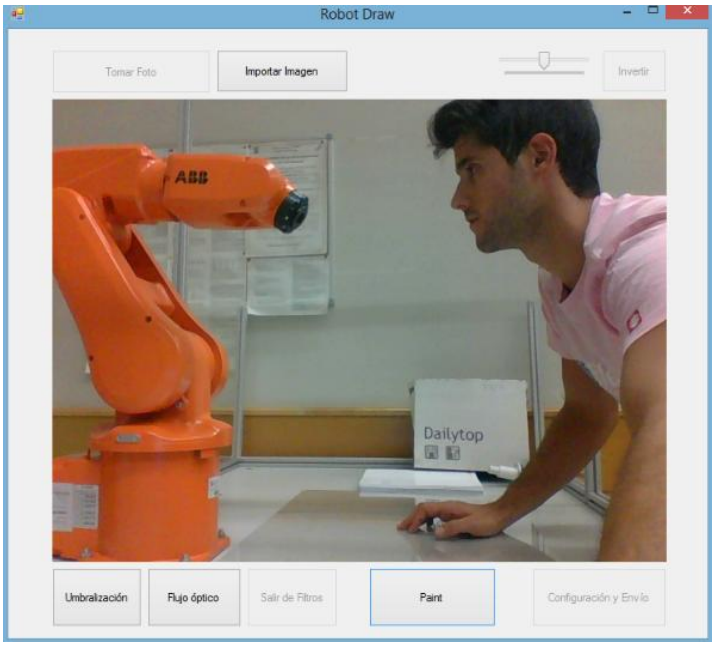


Figura 8.30 Resultados – Imagen adquirida por web cam en vista previa

Esta es la imagen que se ha utilizado como ejemplo para realizar la representación del caso "Imagen adquirida desde la cámara web". Esta imagen será procesada mediante los pasos detallados en capítulos anteriores hasta llegar a los resultados que a continuación se presentan.

Se presentan dos casos para cada tipo de representación de líneas y puntos. El primer caso con una separación en el muestreo de 4mm y el segundo caso con una separación de 10mm, de esta forma se puede apreciar de forma significativa el cambio. Además se incluye capturas de la ventana configuración en cada caso, con lo que quedan detallados los parámetros de configuración

8.4.3.1.1 Representación imagen importada mediante líneas

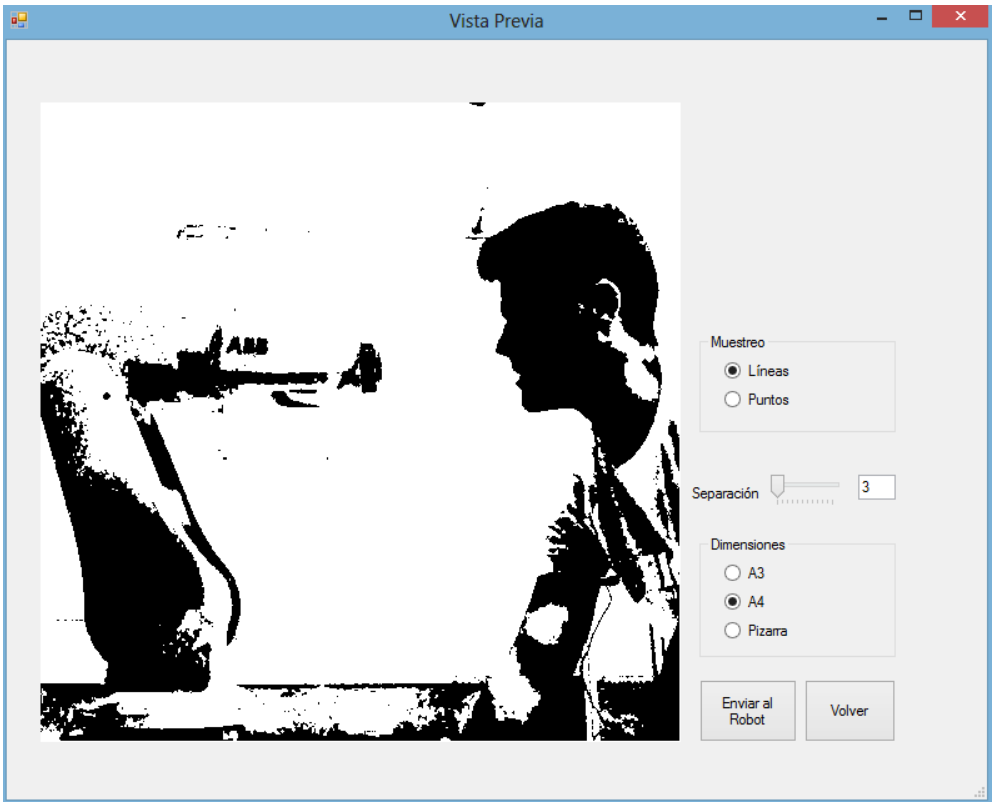


Figura 8.31 Imagen captada por la web cam – vista previa

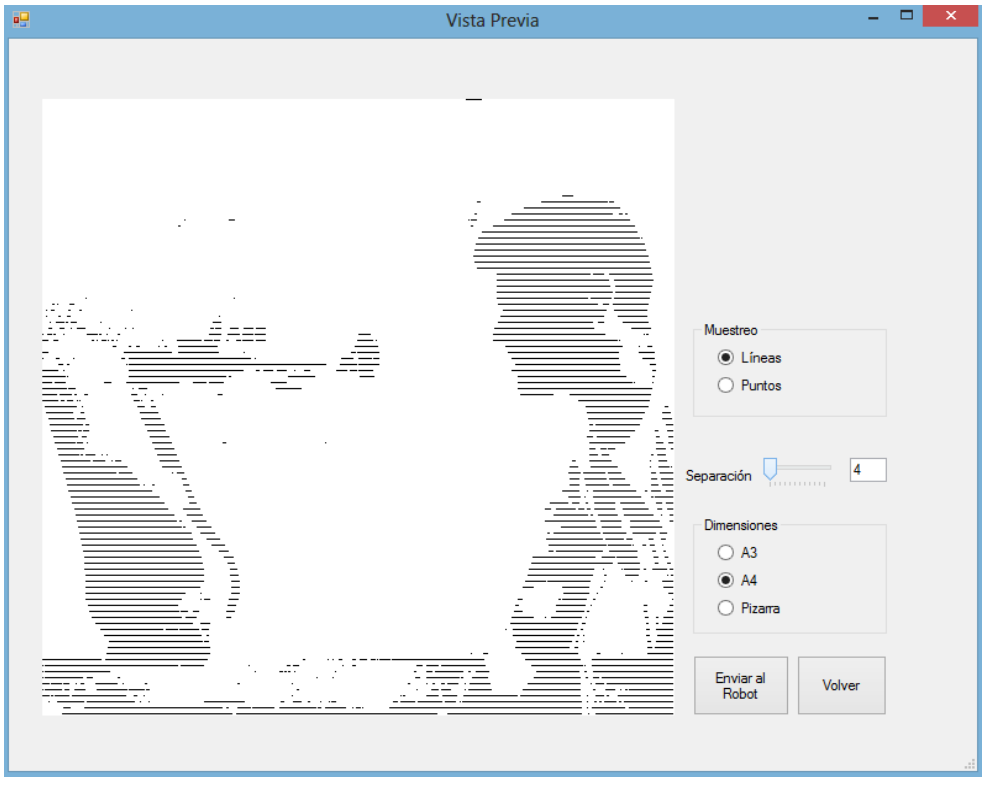


Figura 8.32 Resultados – Imagen captada por la cámara web – vista previa - Líneas

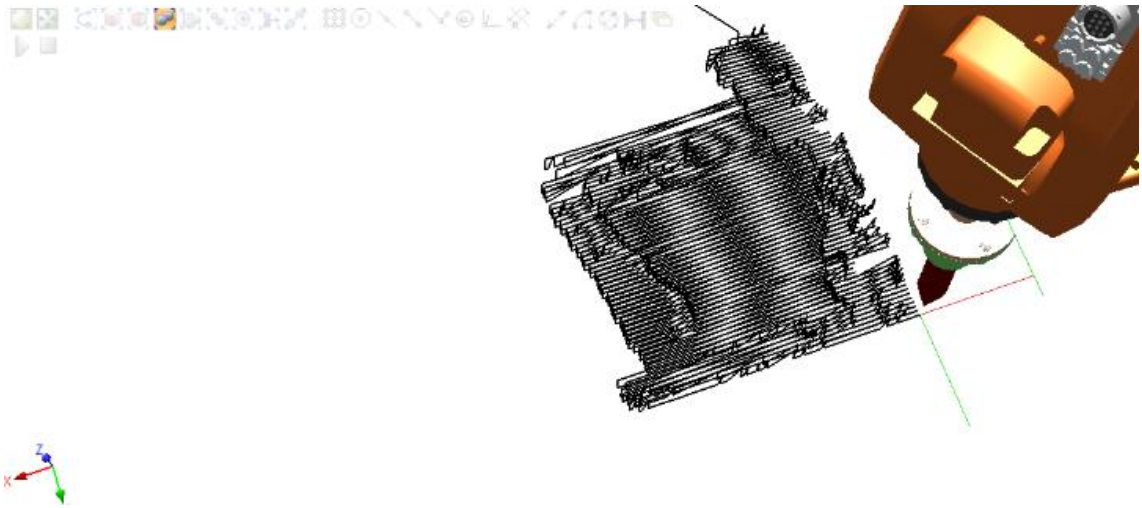


Figura 8.33 Resultados – Imagen captada por la cámara web vista superior – Líneas 4 mm

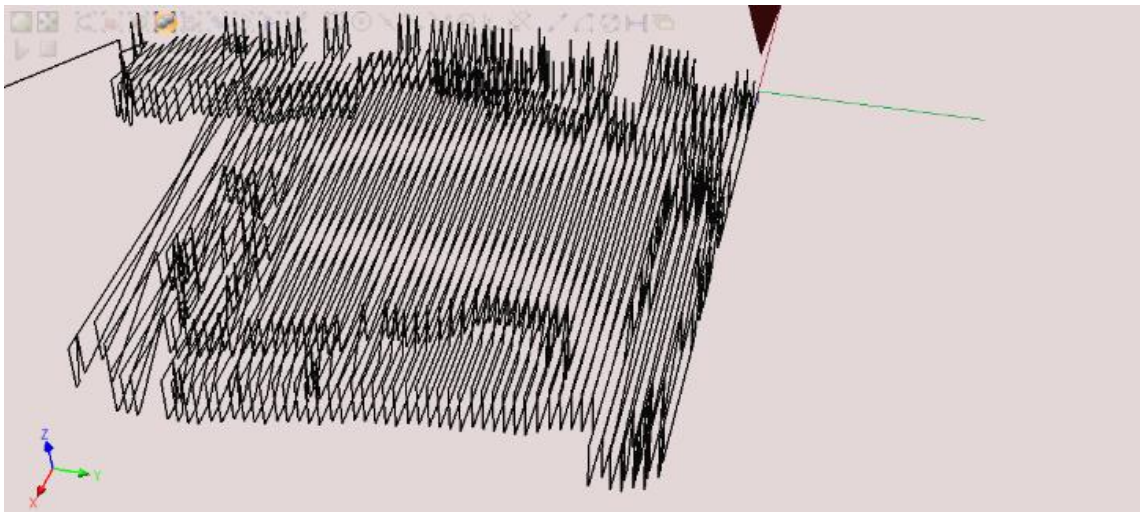


Figura 8.34 Resultados – Imagen captada por la cámara web vista superior angulada – Líneas 4 mm

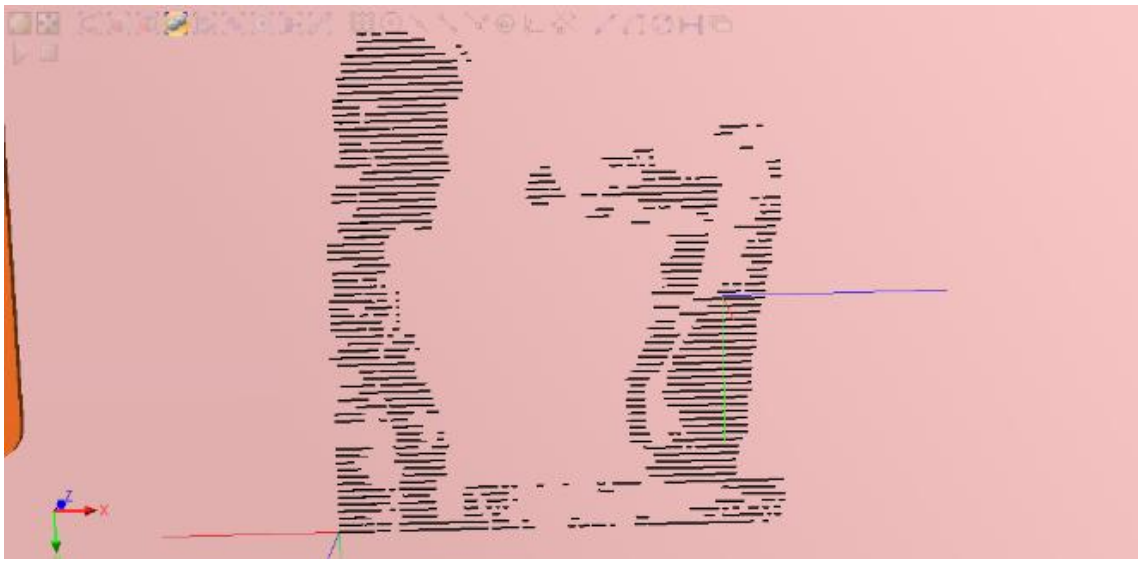


Figura 8.35 Resultados – Imagen captada por la cámara web vista inferior frontal – Líneas 4 mm

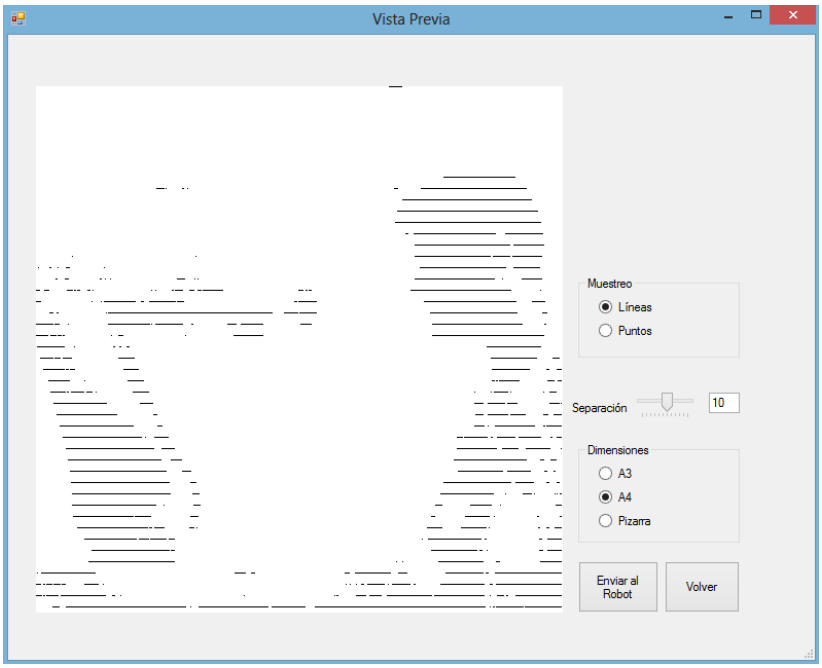


Figura 8.36 Resultados – Imagen captada por la cámara web en ventana de configuración – Líneas 10 mm

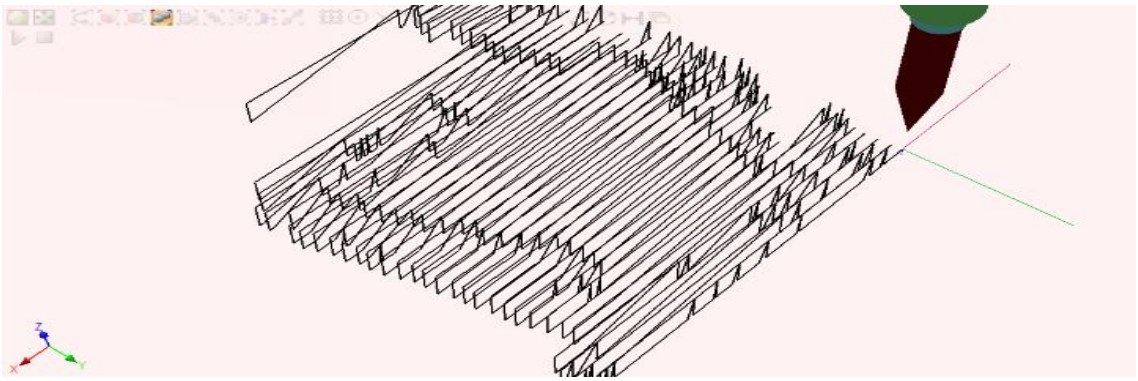


Figura 8.37 Resultados – Imagen captada por la cámara web vista inferior angulada - Líneas 10 mm

Figura 8.38 Resultados – Imagen captada por la cámara web vista inferior frontal - Líneas 10 mm

8.4.3.1.2 Representación imagen importada mediante puntos

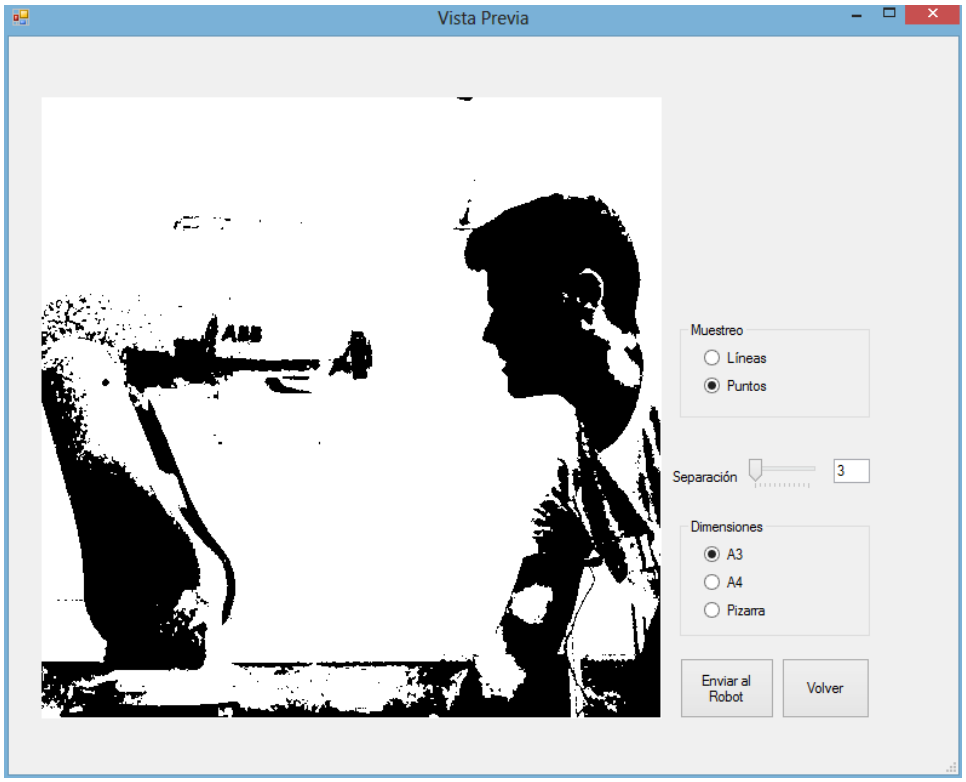


Figura 8.39 Resultados – Imagen captada por la cámara web – vista previa

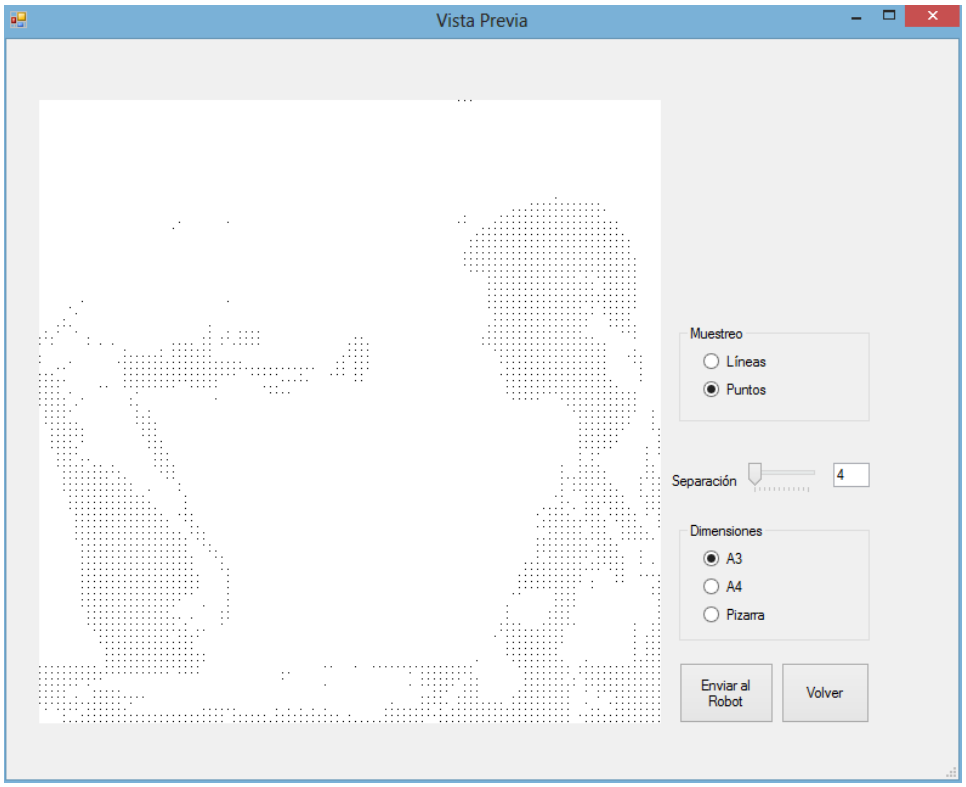


Figura 8.40 Resultados – Imagen captada por la cámara web en vista previa– Puntos 4 mm

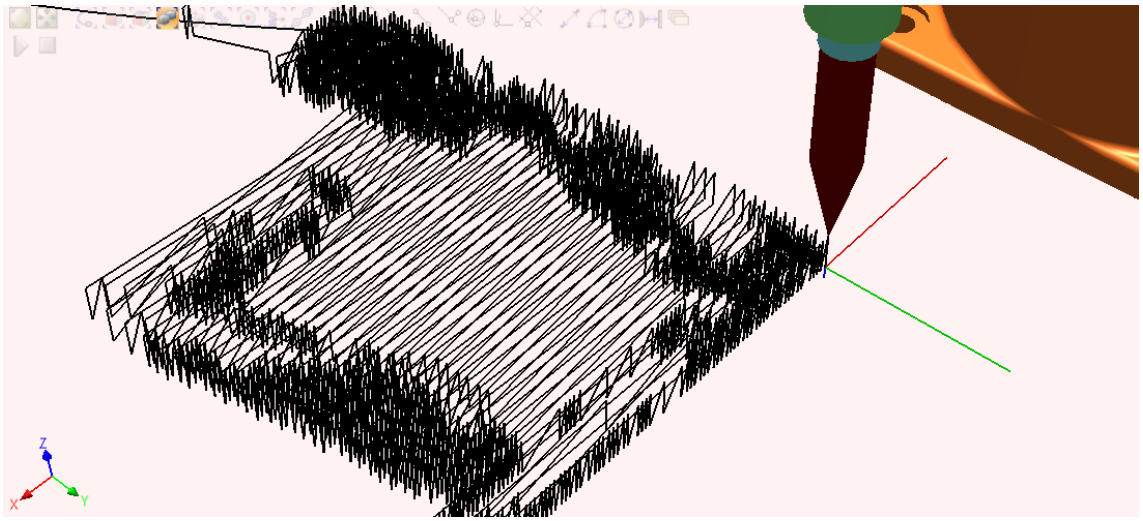


Figura 8.41 Resultados – Imagen captada por la cámara web vista superior– Puntos 4 mm

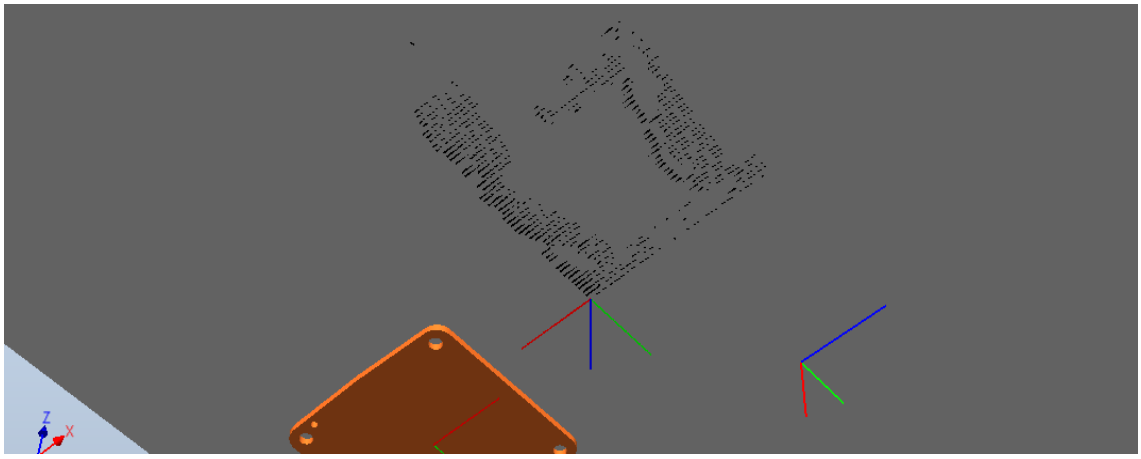


Figura 8.42 Resultados – Imagen captada por la cámara web vista angulada – Puntos 4 mm

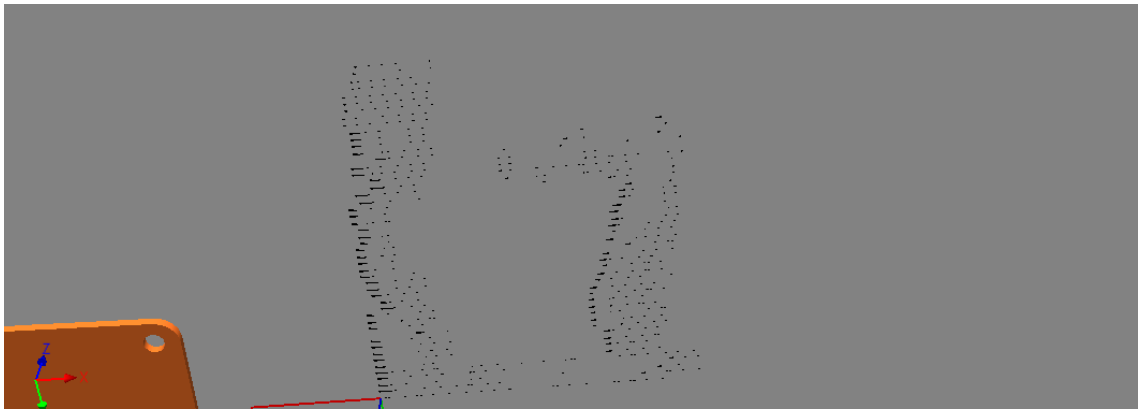


Figura 8.43 Resultados – Imagen captada por la cámara web vista inferior– Puntos 4 mm

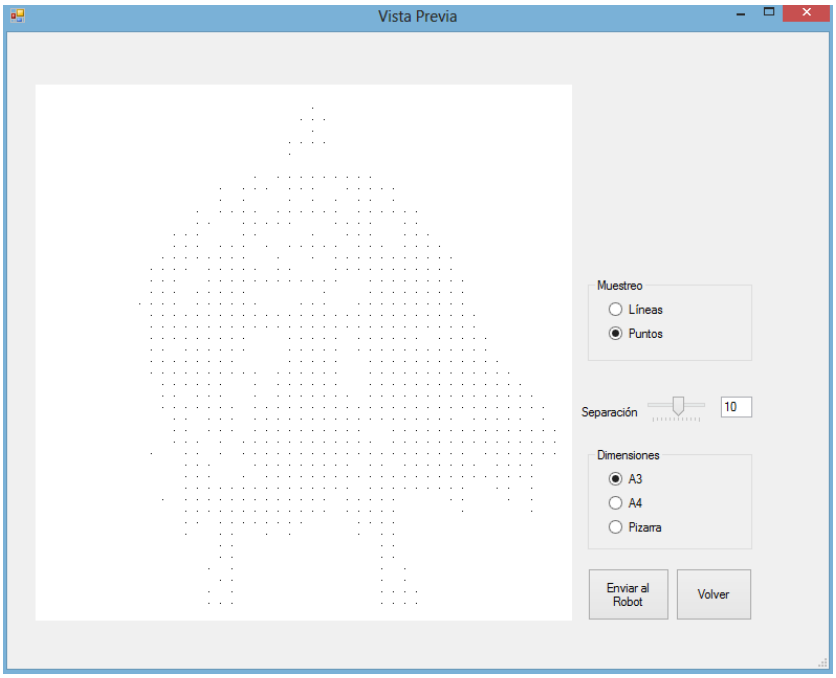


Figura 8.44 Resultados – Imagen captada por la cámara web en vista previa– Puntos 10 mm

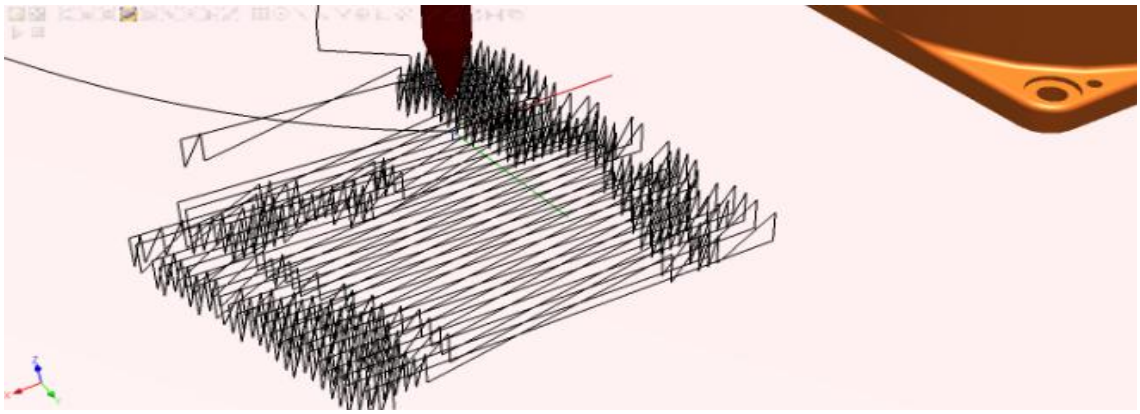


Figura 8.45 Resultados – Imagen captada por la cámara web en vista angulada– Puntos 10 mm

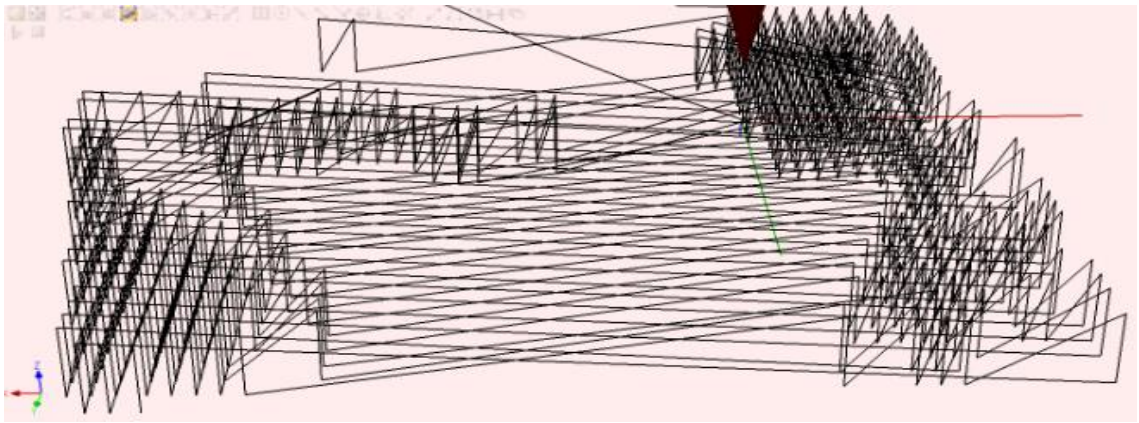


Figura 8.46 Resultados – Imagen captada por la cámara web en vista superior– Puntos 10 mm

8.5 Representación en el brazo ABB –IRB 120

El otro caso que se presenta es la implementación de la aplicación en la estación real del robot. Esta está compuesta por el brazo robótico ABB – IRB 120, el controlador IR5S y por el mando flex pendant. Las fotos que a continuación se presentan son tomadas directamente de la estación con la que se ha realizado el proyecto.

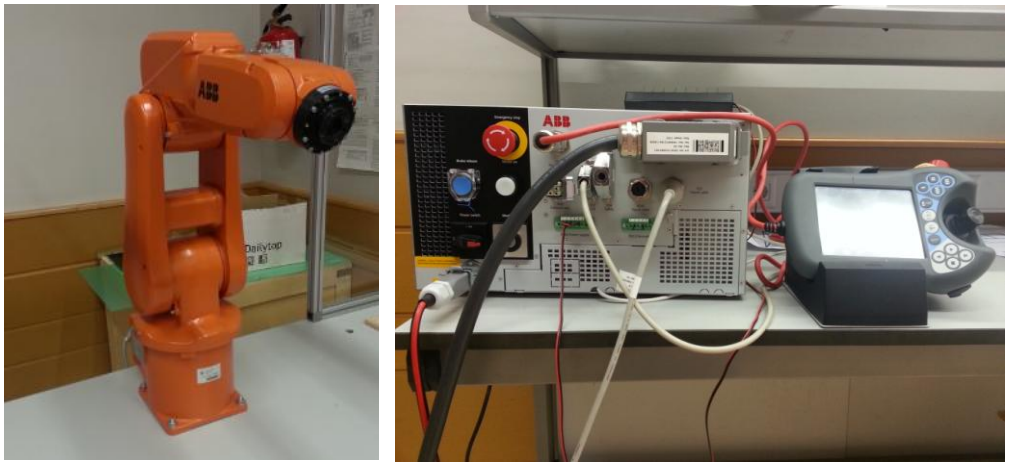
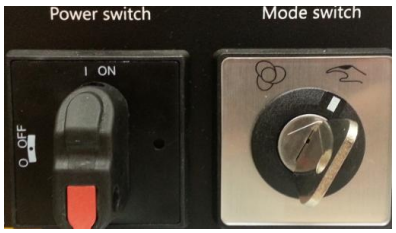


Figura 8.47 Hardware – Estación del robot Abb IRB - 120

8.5.1 Manual de usuario

Antes de mostrar los casos prácticos correspondientes a este apartado, se va a realizar un pequeño manual con el que el usuario sea capaz de representar estos mismos ejemplos de manera autónoma.



Para comenzar, se debe actuar sobre estas llaves. La primera llave a ON, para encender la estación. Y la segunda llave indica el modo de funcionamiento, ya sea manual o automático. Es recomendable dejarlo en modo manual tal y como viene representado en la imagen.

Una vez conectada la estación del robot, se pasa a trabajar con el mando Flex Pendant. Lo primero que se debe hacer, si no está hecho ya, es cargar el programa en la estación, para ello se puede conectar una memoria externa en el mando. Una vez conectado:

- Flex Pendant Explorer→ Copiar y pegar el programa en la carpeta en la que se trabaje→ Datos de programa → Cargar el programa → Puntero a main.

Figura 8.48 Interface principal Flex Pendant

Para acceder a los directorios indicados en el párrafo anterior basta con pulsar el icono de **ABB** de la esquina superior izquierda.

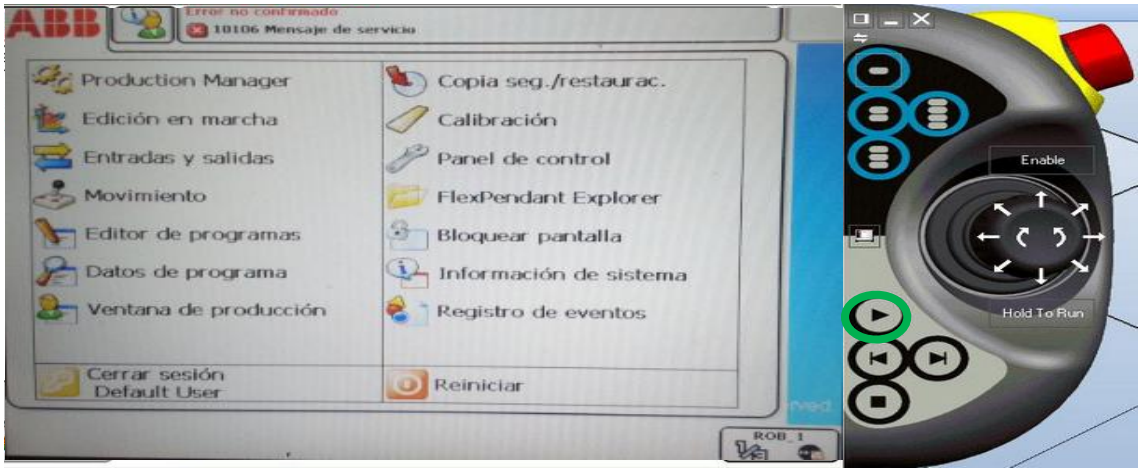


Figura 8.49 Ventana navegación Flex Pendant

Una vez cargado el programa, con las instrucciones indicadas, se presiona el botón del "hombre muerto" y se pulsa el botón **Start.**

Es fundamental resaltar que, de los tres manuales de usuario facilitados en este documento, los dos últimos son independientes y atienden a objetivos distintos (representación virtual – representación real). Sin embargo complementan al primero, que es la guía de la aplicación.

Importante: antes de hacer clic en el botón "Connect" de la ventana "Position Control Panel", primero se ha de llevar a cabo los pasos detallados en este apartado.

8.5.2 Resultados prácticos obtenidos

En este apartado se detallan los resultados prácticos obtenidos mediante la estación de trabajo real del robot. Las imágenes adjuntadas han sido escaneadas de las láminas de dibujo utilizadas para la representación.

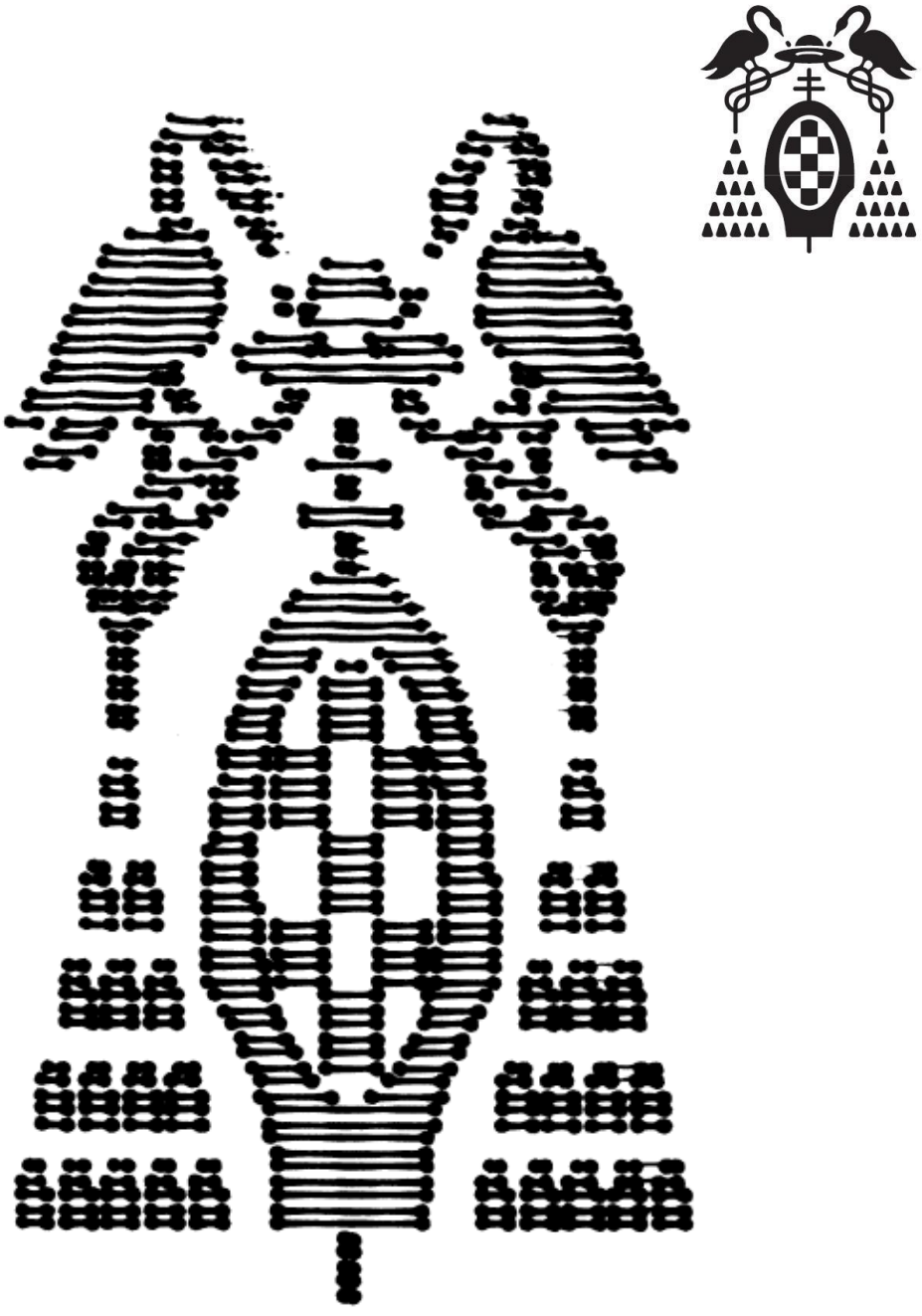


Figura 8.50 Resultados – Imagen representada. 1

Fuente	**Imagen del equipo**	Dimensión	**A3**	Tipo de muestreo	**Líneas**
Separación	**4 mm**	Tipo de imagen	**Escudo Universidad de Alcalá**		

Figura 8.51 Resultados – Imagen representada. 2

Fuente	**Imagen del equipo**	Dimensión	**A3**	Tipo de muestreo	**Líneas**
Separación	**4 mm**	Tipo de imagen	**Paisaje África - Elefantes**		

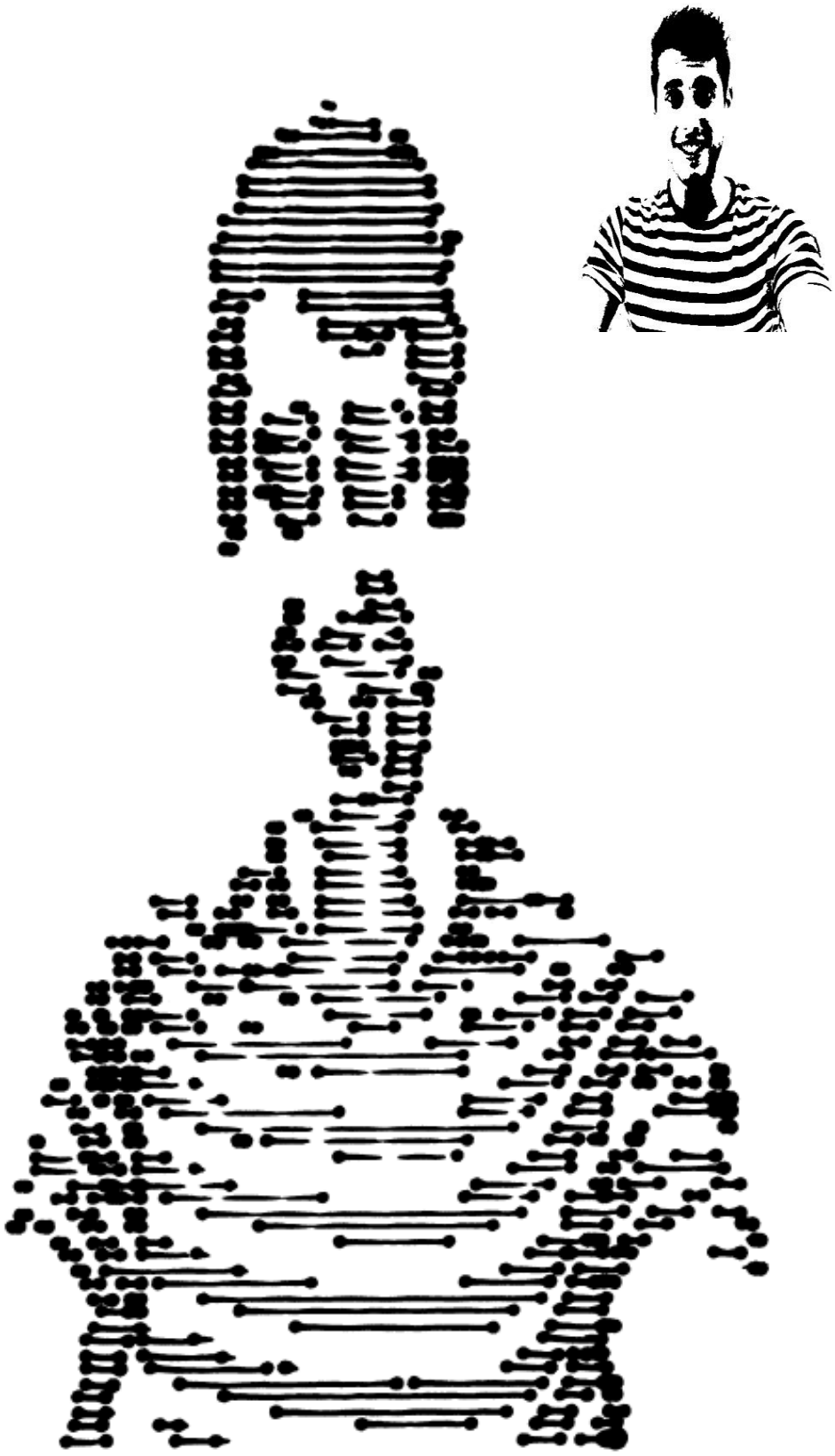


Figura 8.52 Resultados – Imagen representada. 3

Fuente	**Cámara Web**	Dimensión	**A3**	Tipo de muestreo	**Líneas**
Separación	**4 mm**	Tipo de imagen	**Retrato**		

Figura 8.53 Resultados – Imagen representada. 4

Fuente	**Imagen del equipo**	Dimensión	**A3**	Tipo de muestreo	**Líneas**
Separación	**4 mm**	Tipo de imagen	**Roma - Panteón**		

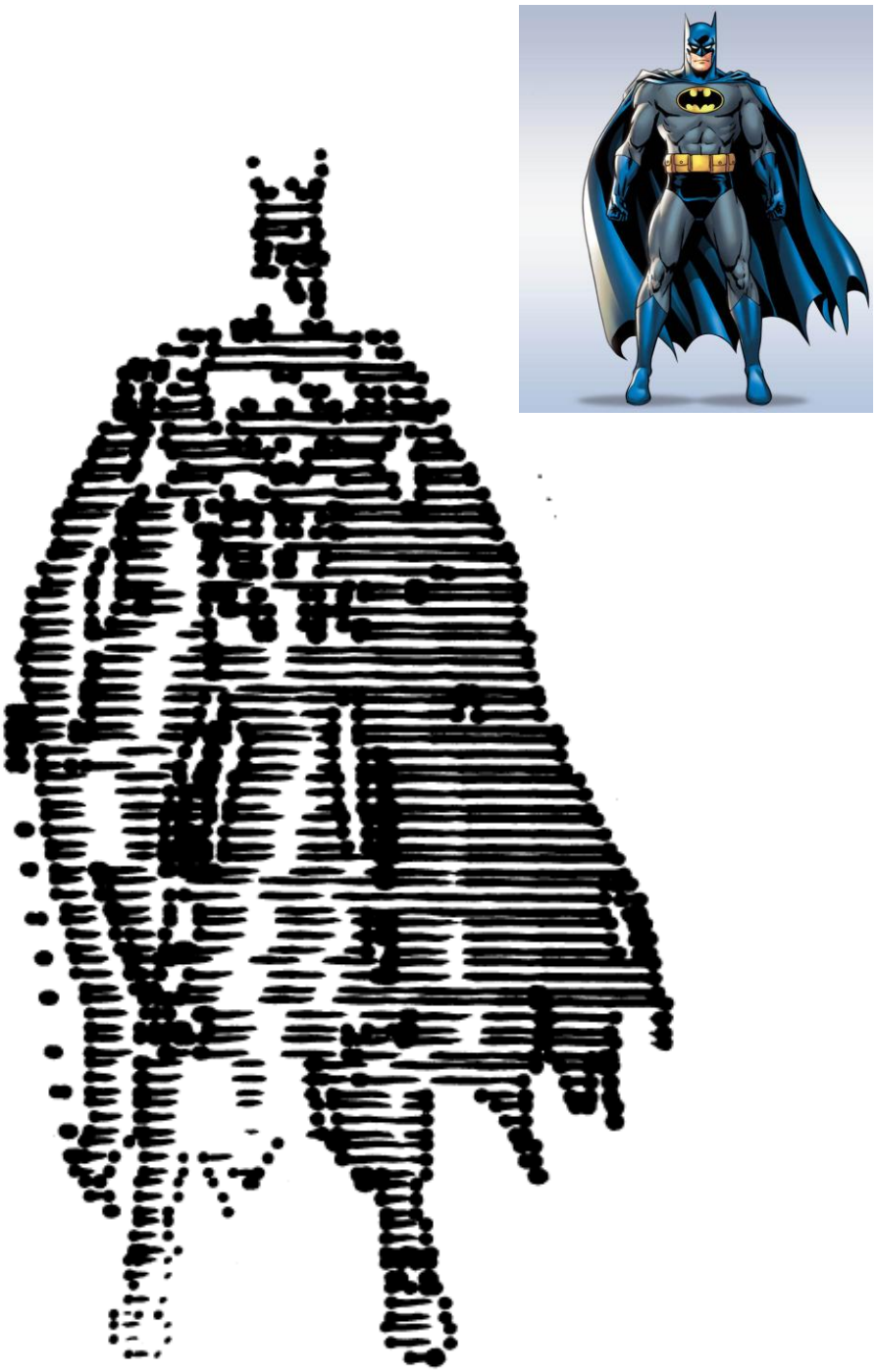


Figura 8.54 Resultados – Imagen representada. 5

Fuente	**Imagen del equipo**	Dimensión	**A4**	Tipo de muestreo	**Líneas**
Separación	**4 mm**	Tipo de imagen	**Dibujo - Batman**		

Figura 8.55 Resultados – Imagen representada. 6

Fuente	**Imagen del equipo**	Dimensión	**A4**	Tipo de muestreo	**Líneas**
Separación	**4 mm**	Tipo de imagen	**Roma - Panteón**		

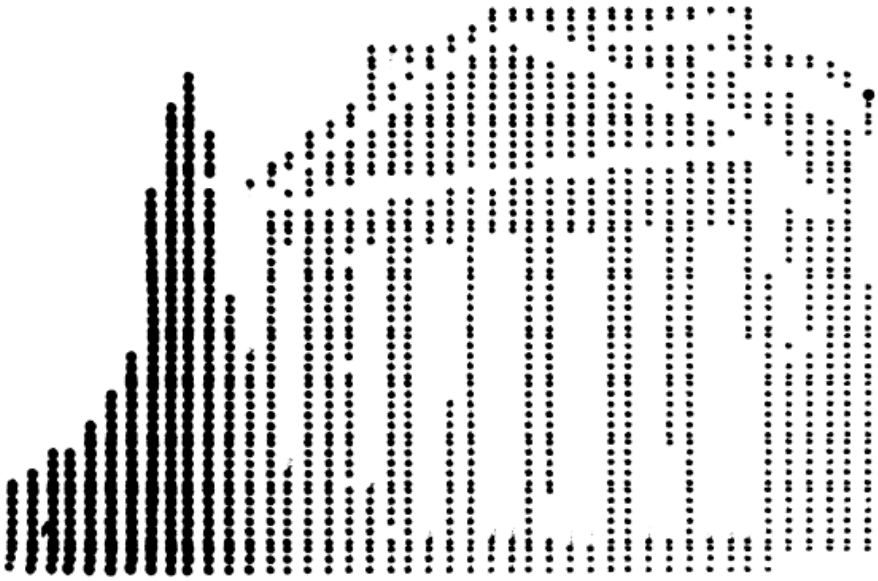


Figura 8.56 Resultados – Imagen representada. 7

Fuente	**Imagen del equipo**	Dimensión	**A4**	Tipo de muestreo	**Puntos**
Separación	**8 mm**	Tipo de imagen	**Roma - Panteón**		

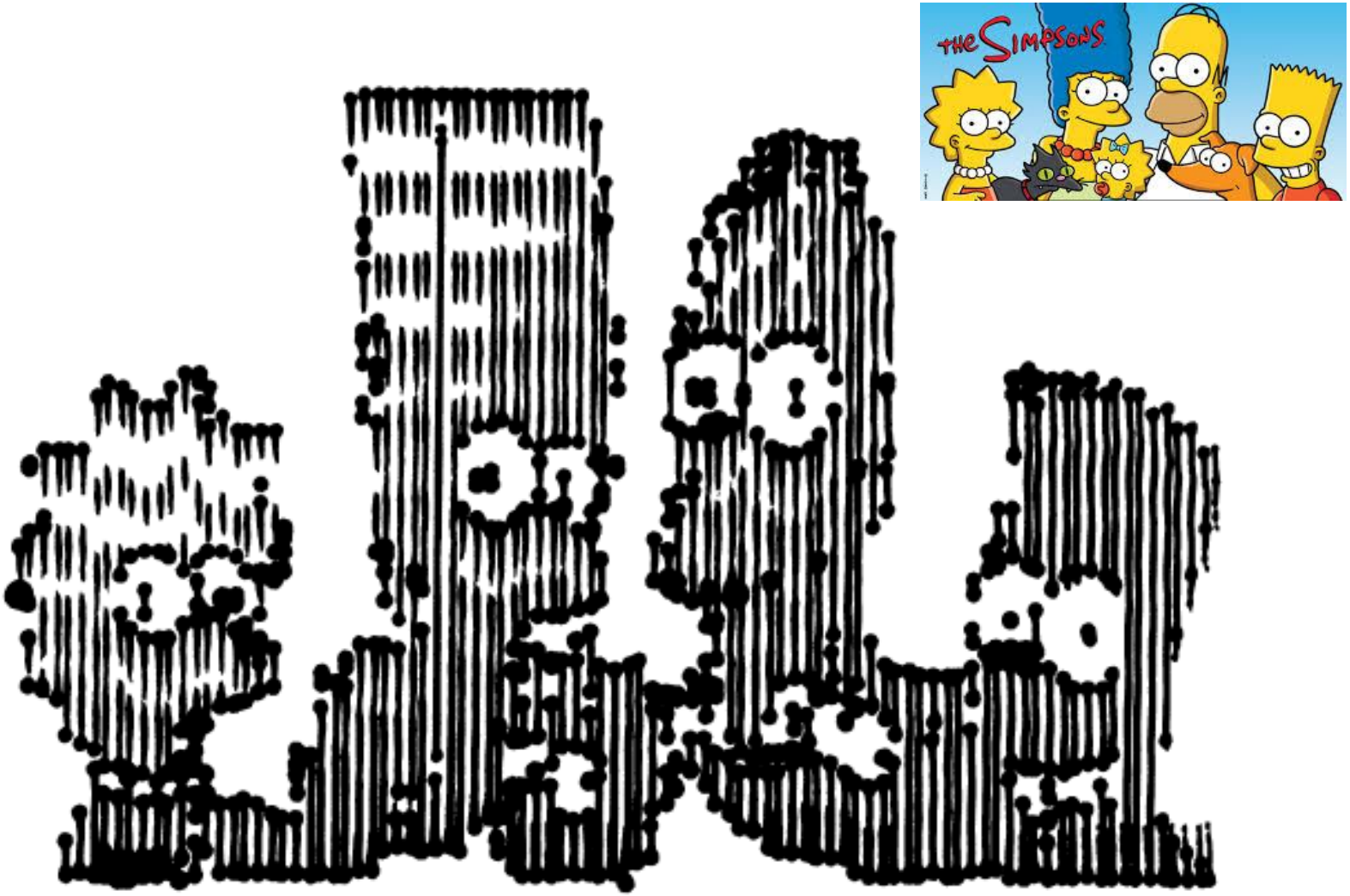


Figura 8.57 Resultados – Imagen representada. 8

Fuente	**Imagen del equipo**	Dimensión	**A4**	Tipo de muestreo	**Líneas**
Separación	**4 mm**	Tipo de imagen	**Dibujo - Simpsons**		

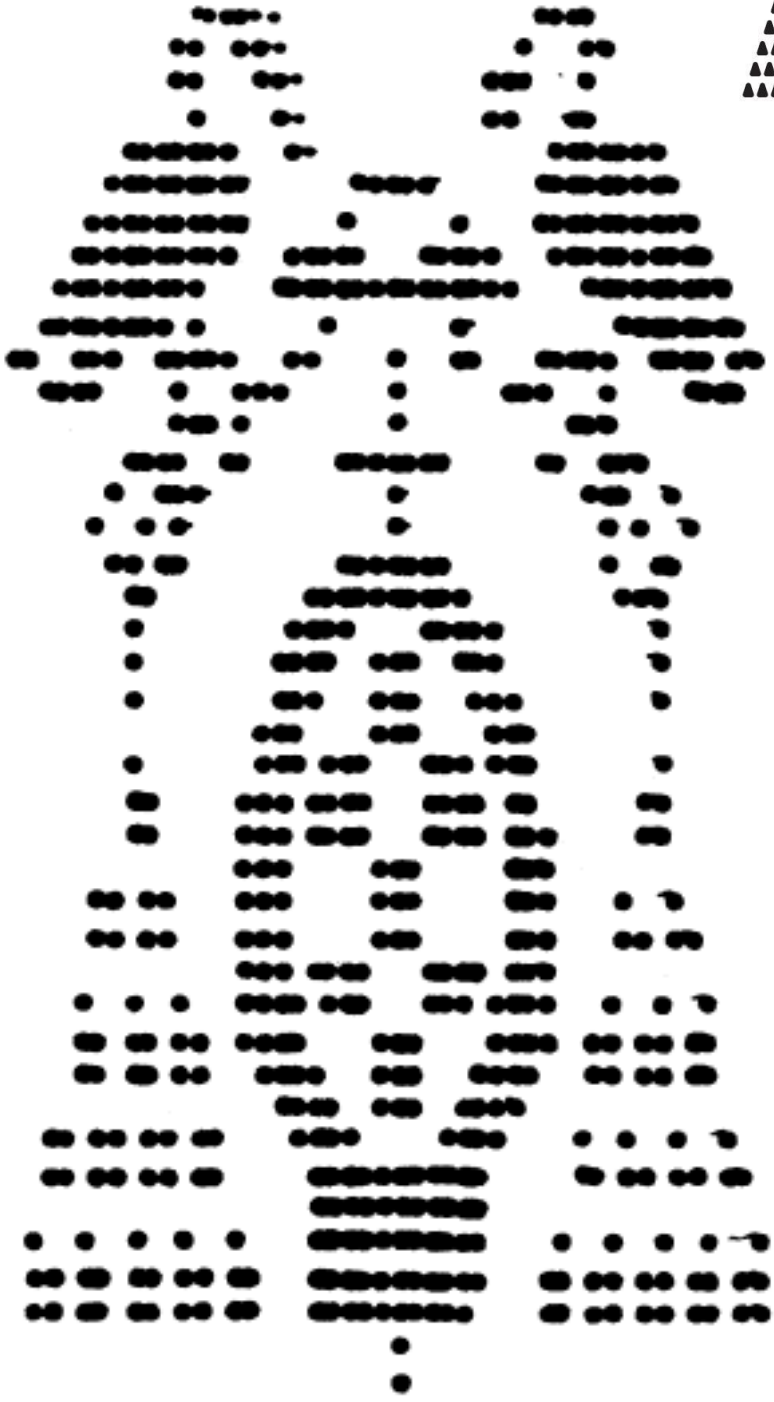


Figura 8.58 Resultados – Imagen representada. 9

Fuente	**Imagen del equipo**	Dimensión	**A4**	Tipo de muestreo	**Puntos**
Separación	**8 mm**	Tipo de imagen	**Escudo Universidad de Alcalá**		

Figura 8.59 Resultados – Imagen representada. 10

Fuente	**Paint**	Dimensión	**A3**	Tipo de muestreo	-
Separación	-	Tipo de imagen	**Trazo libre**		

9 Pliego de condiciones

En este capítulo se detallan las características de las herramientas utilizadas.

9.1 Hardware

1. Portátil Acer Aspire V
    - Procesador: Intel® Core™ i5
    - RAM: 8 GB
    - Tarjeta gráfica : NVIDIA GeForce 720M

2. Robot Industrial ABB-IRB120
    - Peso: 25kg
    - Altura: 580 mm
    - Carga soportada: 3kg (hasta 4 Kg con muñeca vertical)
    - Controlador: IRC5 Compact

9.2 Software

1. Windows 8.1 © Microsoft Corporation.
2. ABB RobotStudio
3. Microsoft Visual C# Express
4. Paint
5. Microsoft Office Home and Student 2010

10 Planos

En este capítulo se detallarán las funciones más importantes implementadas en la aplicación y su flujo de trabajo. Las funciones no están completas, únicamente se detallan las instrucciones básicas.

10.1 Inicio de adquisición mediante Web Cam

```
Private void comenzarCaptura ()
      […]

         Captura.start();


      […]

private void ProcesarFrame(object
sender, EventArgs e)ç

      […]

         imageBox1.Image =
captura.RetrieveBgrFrame();
         return;

      […]
```

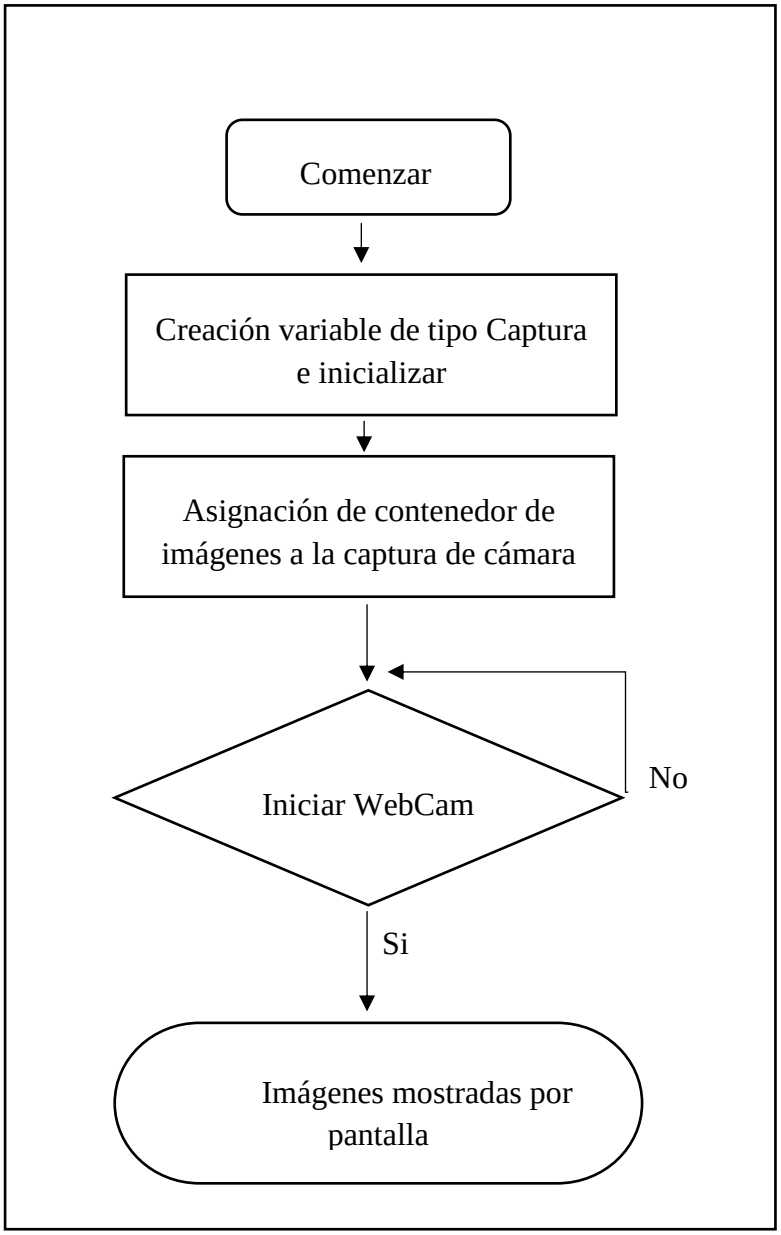


Figura 10.1 Diagrama de flujo – Iniciar captura WebCam

10.2 Proceso de Umbralización

```
private Image<Gray, byte> MiBW(Image<Gray, byte> I)
    […]
        return I.ThresholdBinary(new Gray(val), new Gray(255));
    […]

private void ProcesarFrame(object sender, EventArgs e)
    […]
            imageBox.Image = binarizador(captura.RetrieveGrayFrame()).Not();

            imageBox.Image = binarizador(captura.RetrieveGrayFrame());
    […]

private void thres_Click(object sender, EventArgs e)
    […]
        binarizador = MiBW;
    […]
```

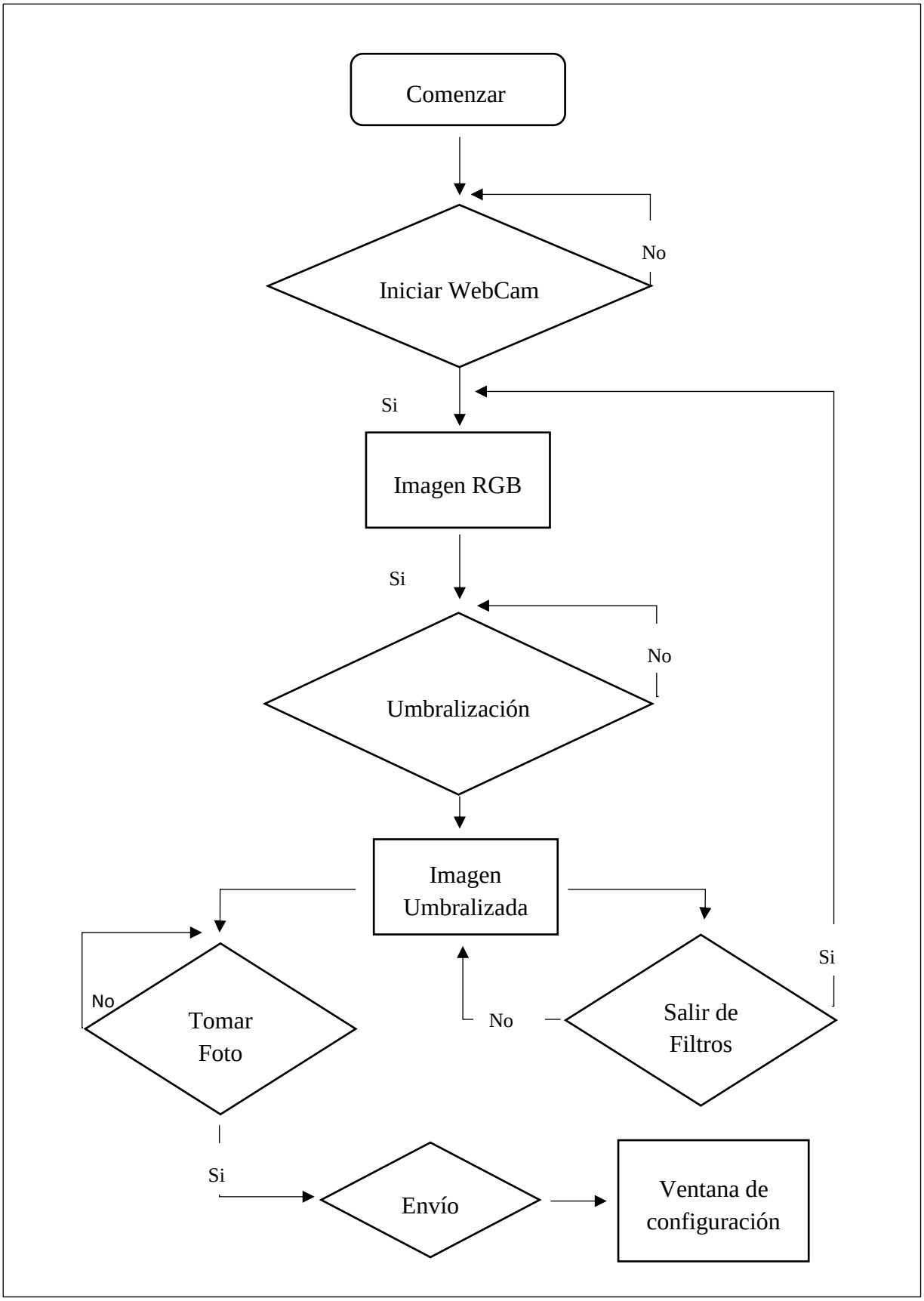


Figura 10.2 Diagrama de flujo – Proceso de umbralización

10.3 Proceso de Flujo óptico

```
pivate Image<Gray, byte> MiOpticalFlow(Image<Gray, byte> I)
    […]
        OpticalFlow.HS(I, J, false, flowX, flowY, 1, new MCvTermCriteria(10));
    […]
        Flujo.Draw(new LineSegment2D(p1, p2), new Gray(0), 1);
    […]

private void ProcesarFrame(object sender, EventArgs e)
    […]
          imageBox.Image = binarizador(captura.RetrieveGrayFrame());
    […]

private void flow_Click(object sender, EventArgs e)
    […]
              binarizador = MiOpticalFlow;
    […]
```

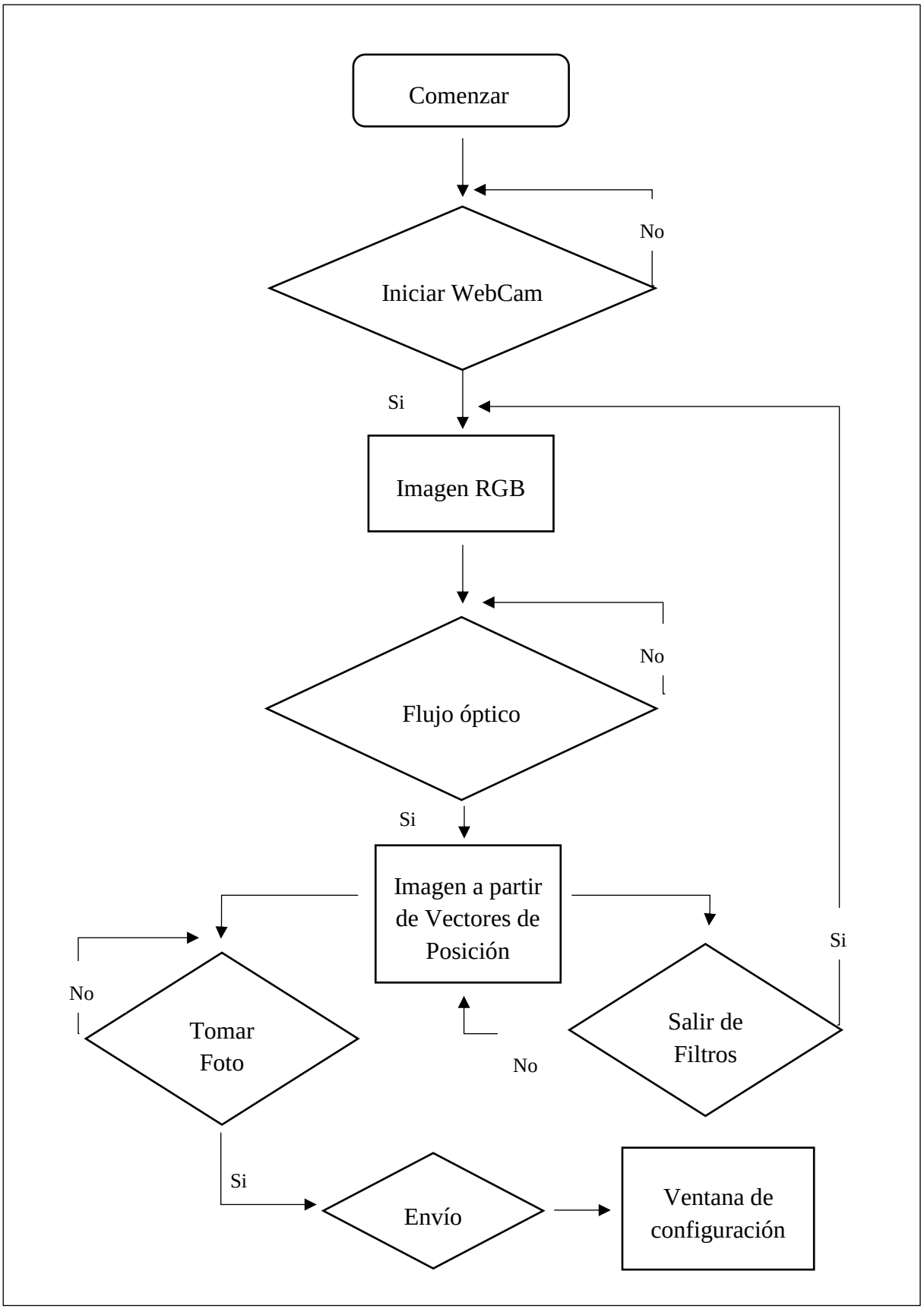


Figura 10.3 Diagrama de flujo – Proceso de flujo óptico

10.4 Proceso Paint

```
private void imageBox_MouseMove(object sender, MouseEventArgs e)
    […]
      I = new Image<Gray, byte>(imageBox.Width, imageBox.Height, new Gray(255));
    […]
        posicion_previa = new Point(e.X, e.Y);
        posicion_actual = new Point(e.X, e.Y);
    […]
     if (e.Button == MouseButtons.Left)
     I.Draw(new LineSegment2D(posicion_actual, posicion_previa), new Gray(0), 1);
    […]
       posicion_previa = posicion_actual;
      imageBox.Image = I;
    […]
```

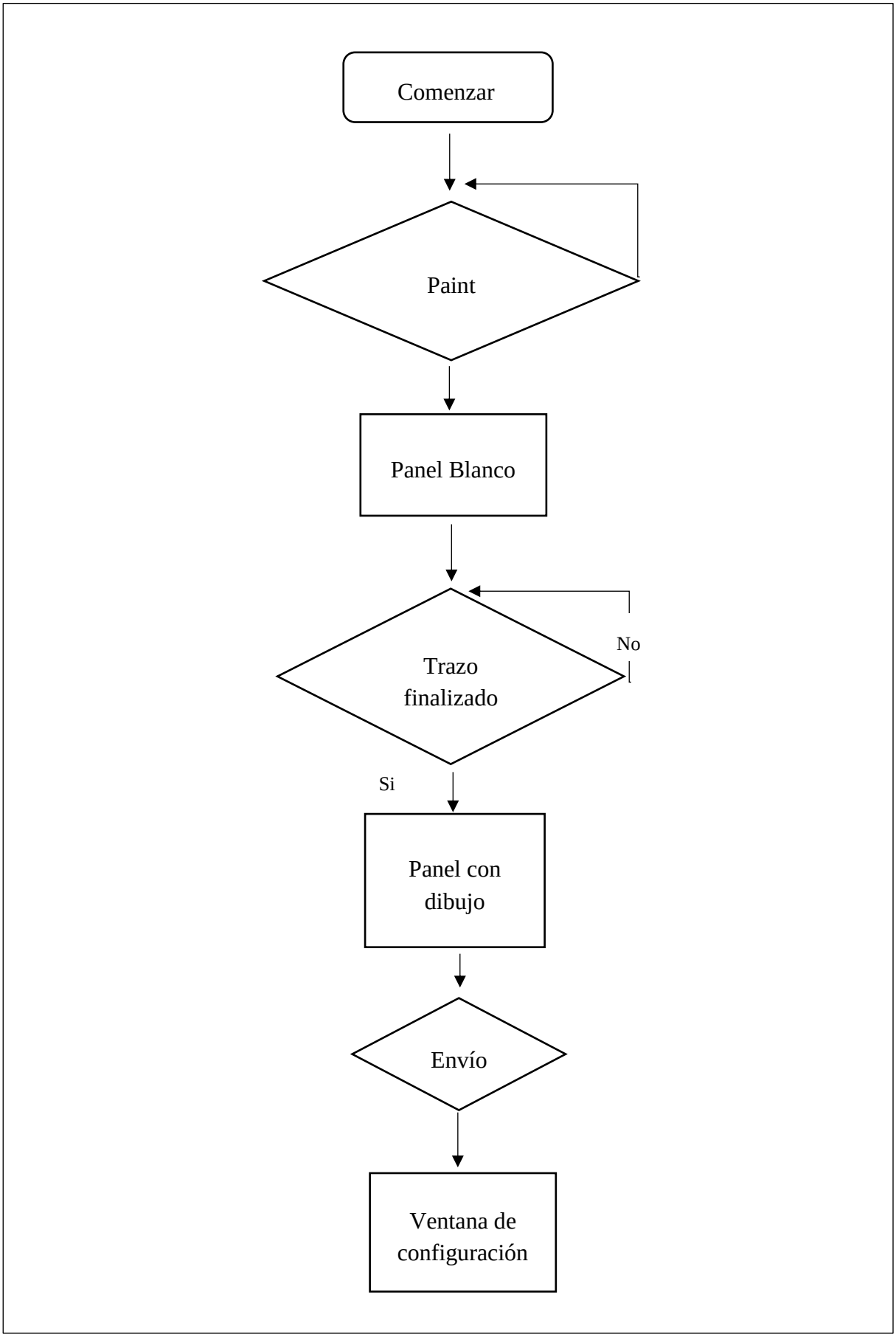


Figura 10.4 Diagrama de flujo – Proceso dibujo Paint y envío

10.5 Diagrama de flujo general

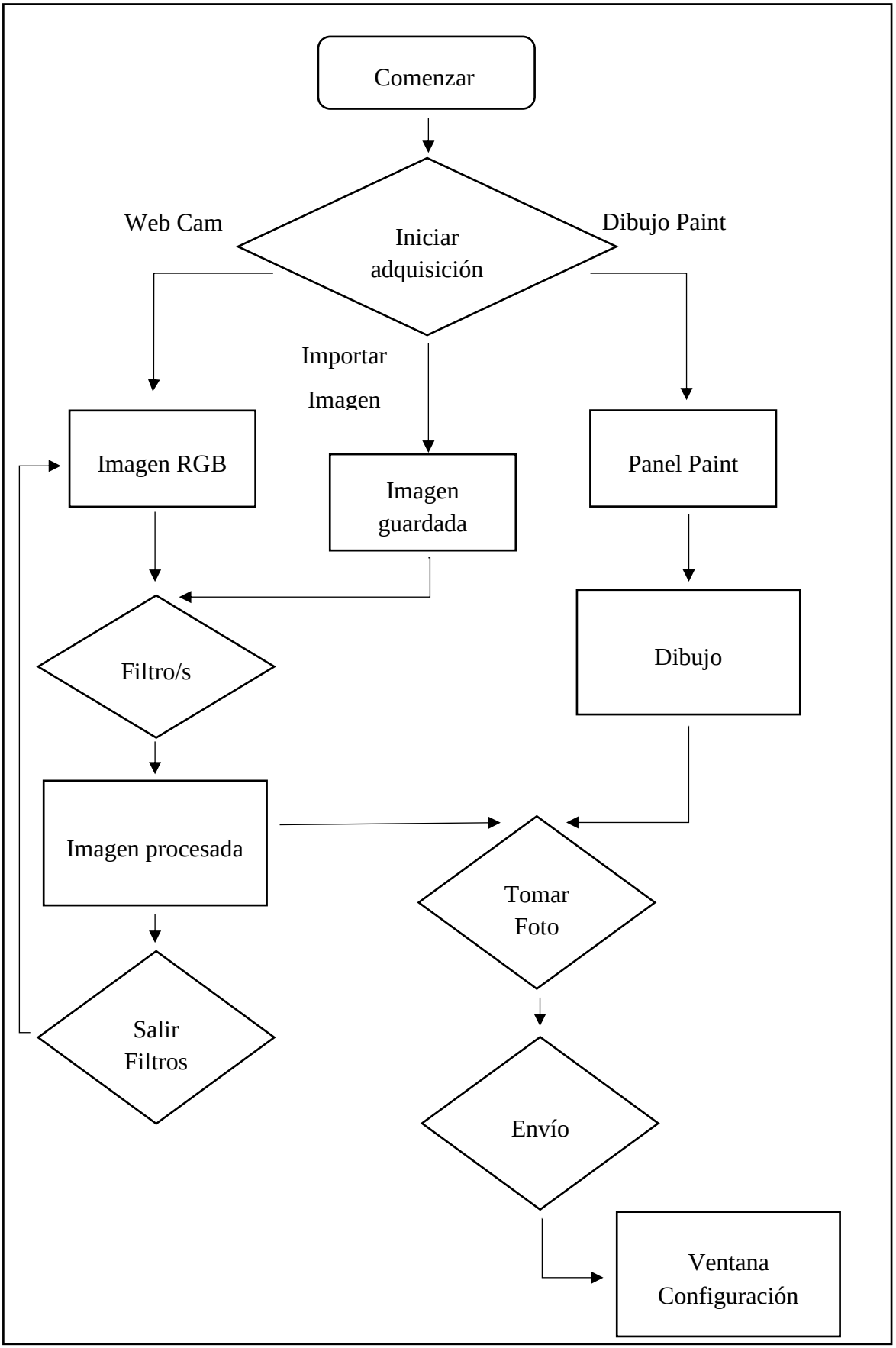


Figura 10.5 Diagrama de flujo – Proceso dibujo Paint y envío

10.6Almacenamiento de información

```
// General

[…]

 private List<Point[]> lineas;
 private List<Point> trazoActual;

 lineas = new List<Point[]>();
 trazoActual = new List<Point>();

 // Flujo Óptico

        Point[] segmento= new Point[2];
        lineas.Add(segmento);


 // Umbralización – Lineas

[…]

      Point[] linea = new Point[2];
      Image<Gray, byte> I = (Image<Gray, byte>)imageBox.Image;
      I = I.Resize(imageBox.Width, imageBox.Height,
INTER.CV_INTER_CUBIC).ThresholdBinary(new Gray(127), new Gray(255));

        linea[0] = new Point(x, y);

        linea[1] = new Point(x, y);

        lineas.Add(linea);

        linea = new Point[2];

[…]
```

```
[…]

if (haylinea)
            lineas.Add(linea);
            linea = new Point[2];

[…]
```

// Umbralización – Puntos

```
[…]

Image<Gray, byte> I = (Image<Gray, byte>)imageBox.Image;
I = I.Resize(imageBox.Width, imageBox.Height,
INTER.CV_INTER_CUBIC).ThresholdBinary(new Gray(127), new Gray(255));
            Point[] linea = new Point[1];
            linea[0] = new Point(x, y);
            lineas.Add(linea);

[…]
```

```
// Paint

    […]

if (e.Button == MouseButtons.Left)

        I.Draw(new LineSegment2D(posicion_actual, posicion_previa), new
Gray(0), 1);

        addPunto(posicion_actual);

    […]

    private void addPunto(Point posicion_actual)

        trazoActual.Add(posicion_actual);

    […]

      Point p = trazoActual[trazoActual.Count() - 1];
      Point d = new Point(posicion_actual.X - p.X, posicion_actual.Y - p.Y);

      Int32 dist = d.Y * d.Y + d.X * d.X;

      if (dist > 4)

    […]

        trazoActual.Add(posicion_actual);

    […]
```

```
// Envío de Información

private void imageSendBoton_Click(object sender, EventArgs e)

    […]


        foreach (Point[] trazo in lineas)

    […]

            posZ = ZARRIBA;
            sendConvertedPoint(trazo[0]);
            posZ = ZABAJO;

    […]


            foreach (Point p in trazo)
    […]

            sendConvertedPoint(p);

    […]


    private void sendConvertedPoint(Point p)

    […]

       xc =(int)Math.Round(xMax - p.X * (xMax - xMin) / (float)anchura);
       yc = (int)Math.Round(yMin + p.Y * (yMax - yMin) / (float)altura);

       Point pt = new Point(xc, yc);

       sendPoint(pt);

    […]
```

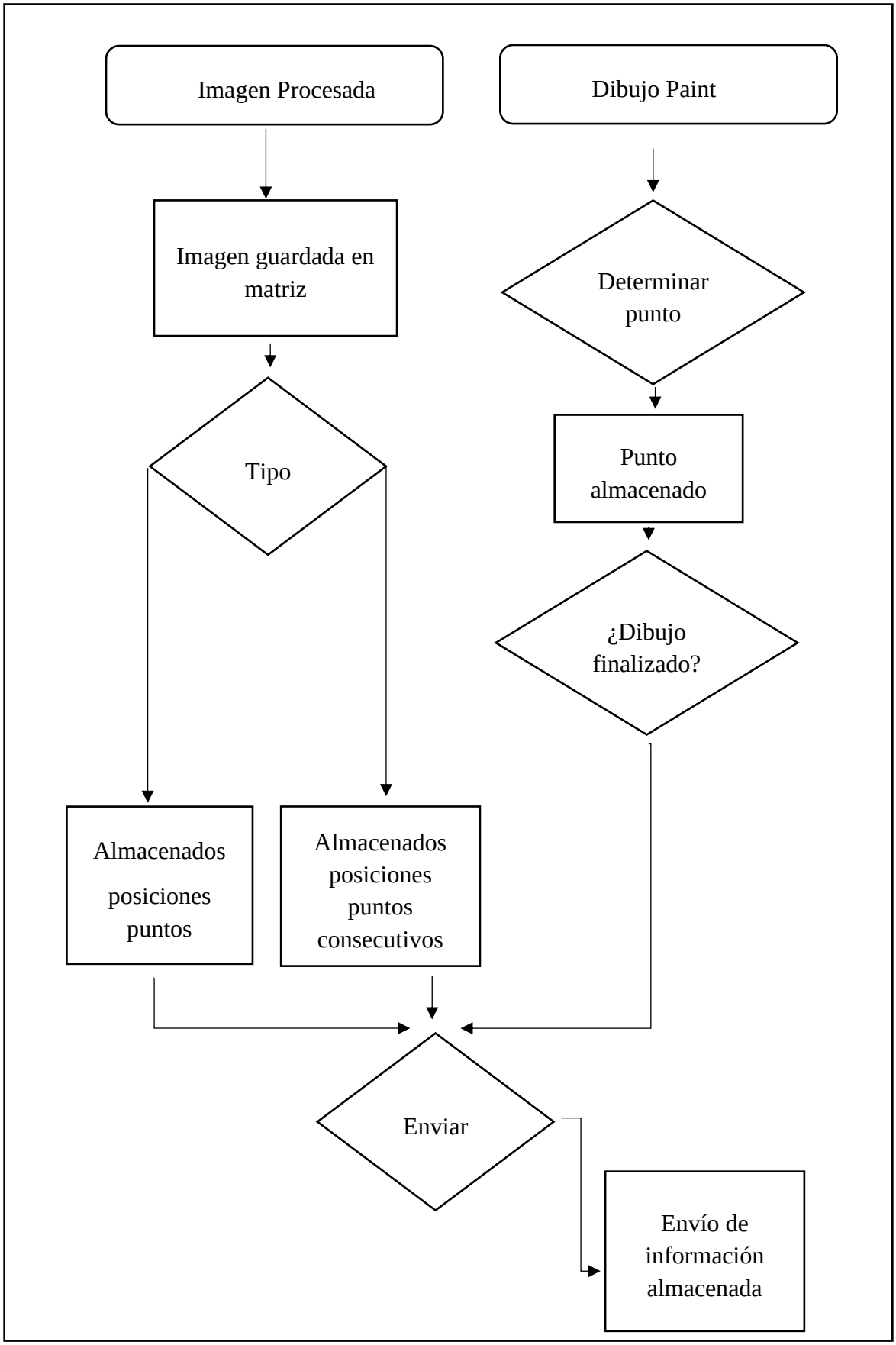


Figura 10.6 Diagrama de flujo – Envío de información

11 Presupuesto

Este capítulo proporciona información detallada acerca de los costes teóricos del desarrollo del proyecto, incluyendo los costes de materiales y las tasas profesionales.

11.1 Costes materiales

Objeto		***Cantidad***	***Precio unidad (€)***	***Precio total (€)***
Hardware	Portátil Acer	1	600€	600€
	ABB-IRB120	1	10954,00	10954,00
Software	Microsoft Windows 8.1	1	0,00	0,00
	RobotStudio	1	0,00	0,00
	Microsoft Visual C# Express	1	0,00	0,00
	Paint	1	0,00	0,00
	Microsoft Office Home and Student 2010	1	100,00	100,00
TOTAL				11654,00

Tabla 11.1 Costes materiales (hardware y software) sin IVA.

11.2 Tasas profesionales

Las tasas profesionales incluyen los costes de realización del proyecto, entendiéndose estos costes como el tiempo dedicado a su desarrollo.

Objeto	*Tiempo (meses)*	*Coste unidad (€)*	*Coste total (€)*
Ingeniería	4	1300 €	5200 €
Escritura	1	800 €	800 €
TOTAL			6000 €

Tabla 11.2 Costes profesionales (hardware y software) sin IVA.

11.3Costes totales

Los costes totales del Proyecto han sido obtenidos sumando los costes materiales y profesionales aplicándose el IVA. Además hay que tener en cuenta los costes de impresión y encuadernación.

Objeto		*Costes totales*
Costes materiales		11654,00 €
Costes profesionales		6000,00 €
Proyecto	Impresión	40 €
	Encuadernación	100 €
Subtotal		17794 €
IVA (21%)		3763 €
TOTAL		21530 €

Tabla 11.3 Costes totales con IVA.

Los costes totales de la implementación del proyecto ascienden a veinte un mil quinientos treinta euros

12 Bibliografía

[1] http://new.abb.com

[2] Enciclopedia del lenguaje C#. Francisco Javier Ceballos

[3] http://www.emgu.com/wiki/index.php/Main_Page

[4] http://sourceforge.net/projects/emgucv/files/latest/download?source=files

[5] http://msdn.microsoft.com/es-es/library/z1zx9t92.aspx

[6] Transparencias asignatura “Visión Artificial”. UAH

[7] http://opencv.org/

[8] http://www.youtube.com/watch?v=vdjoutNR2DQ